

MATLAB GEOMETRY BUILDER AND MLFMA MODELER

By

CHRISTOPHER CARRERO

Master of Science in Electrical Engineering
Oklahoma State University
Stillwater, OK, USA
2011

Submitted to the Faculty of the
Graduate College of
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 2011

COPYRIGHT ©

By

CHRISTOPHER CARRERO

July, 2011

MATLAB GEOMETRY BUILDER AND MLFMA MODELER

Thesis Approved:

Dr. James West

Thesis Advisor

Dr. Chuck Bunting

Dr. James Stine

Dr. Mark E. Payton

Dean of the Graduate College

ACKNOWLEDGMENTS

First and foremost I need to thank God for his constant presence during my countless hours spent performing research, generating and debugging code, creating geometries and models, and writing and re-writing the documentation itself. As I found myself doing more and more work under the cover of night, He was my only companion. I give Him all the glory for the work I have produced. I'd like to thank Dr. West for all his careful explanations of the inner-workings of the pre-existing code and for his quick responses to my inquisitive and long-winded emails. I'd like to thank my wife for her patience in rarely being the center of my attention in our first year of marriage and for her willingness to support my hectic work and study schedule; always providing meals and words of encouragement whenever I would emerge from the office. I'd like to thank my friends and family for all their prayers and support and for putting up with all the missed calls and unanswered messages.

TABLE OF CONTENTS

Chapter		Page
1	Introduction	1
2	Basic Computational Electromagnetic Code Theory	4
2.1	Incident Fields and Induced Currents	4
2.2	Geometry Discretization	4
2.3	Iterative Solvers and Preconditioners	5
2.4	Calculate Currents and Fields	6
2.5	EM Software	6
3	File Structure and Functional Blocks	8
3.1	Directory Hierarchy	10
3.1.1	MLFMA Directory	11
3.1.2	Projects Directories	12
3.1.3	Geometry Directories	13
3.1.4	Model Directories	14
3.2	File Types	15
3.2.1	Interface Geometry Files	16
3.2.2	Delaunay Geometry Files	17
3.2.3	Field Source Files	19
3.2.4	Surface Current Data Files	20
3.2.5	Field Intensity Files	21
3.2.6	Log Files	22

3.3	Functional Blocks	24
3.3.1	Project Loader	25
3.3.2	GUI Workspace	26
3.3.3	Geometry Builder	27
3.3.4	Geometry Modifier	28
3.3.5	Geometry Updater	34
3.3.6	Element Remover	37
3.3.7	Element Joiner	41
3.3.8	System Modeler	43
3.3.9	Data Visualization	46
4	Geometry Modeling	50
4.1	Backscatter Models	50
4.1.1	Sphere	51
4.1.2	Plate	56
4.1.3	Cube	58
4.1.4	Hemispherical Dipole	60
4.2	Radiation Models	63
4.2.1	Dipole	63
4.2.2	Yagi-Uda Antenna	66
4.2.3	Dipole In Spherical Cavity	69
4.2.4	Helical Antenna	70
4.2.5	Dipole Over Finite Ground Plane	72
5	CONCLUSIONS	75
	BIBLIOGRAPHY	80

LIST OF TABLES

Table		Page
1.1	Primary Thesis Goals	3
3.1	Files Common to MLFMA Directory	11
3.2	File Types	15
3.3	Function Block Descriptions	24
3.4	Builder Geometries	28
3.5	Viewer Types	46
4.1	Yagi-Uda Dimensions	66
4.2	Component Geometry Parameters	67
4.3	Helix Parameters	70

LIST OF FIGURES

Figure	Page
3.1 Typical Directory Hierarchy	10
3.2 Projects Folder Contents	12
3.3 Geometry Directories	13
3.4 Model Directory	14
3.5 Interface Geometry File Format	16
3.6 Delaunay Geometry File Format	18
3.7 Field Source File	19
3.8 Add Field Source GUIs	19
3.9 Surface Current Data File Format	20
3.10 Backscatter Field Intensity File Format	21
3.11 Log File Information	22
3.12 Model Log File Information	23
3.13 Project Loader GUI	25
3.14 GUI Workspace	26
3.15 Geometry Builder	27
3.16 Geometry Modifier	28
3.17 Translation in X-Y Plane	29
3.18 Translation in Z	29
3.19 Centroid Rotation in θ	30
3.20 Centroid Rotation in Phi	30
3.21 Rotate Around Origin in θ	31
3.22 Rotate Around Origin in θ then ϕ	31

3.23	Scale Centroid	32
3.24	Scale Origin	33
3.25	Geometry Updater	34
3.26	Original Plate Copied and Shifted 3 Times	35
3.27	Plates Combined to Form Columns	36
3.28	Element Remover	37
3.29	Cartesian Removal of Sphere Elements	38
3.30	Spherical Removal of Retro-reflector Elements	39
3.31	Spherical Removal of Retro-reflector Elements	40
3.32	Element Joiner	41
3.33	Triangular Element Creation	42
3.34	Common Portion of System Modeler GUIs	43
3.35	Backscatter GUI	44
3.36	Source Model GUIs	45
3.37	GUI Workspace, Completed Models, and Surface Currents	46
3.38	Geometry Viewer	47
3.39	Surface Current Viewer	48
3.40	Field Pattern Viewer	49
4.1	Loading the GUI from the MATLAB Command Line	51
4.2	Create Project	51
4.3	Toolbar Element	52
4.4	geoconvert.m Progress Bar	52
4.5	Sphere Geometry	53
4.6	Models and Currents	54
4.7	Sphere Surface Currents	54
4.8	Sphere Backscatter Pattern	55
4.9	Plate Build	56

4.10 Plate Surface Currents	57
4.11 Plate Backscatter Pattern	57
4.12 Cube Build	58
4.13 Plate Surface Currents	59
4.14 Cube Backscatter Pattern	59
4.15 Hemisphere Components	60
4.16 Element Creation	61
4.17 Finished Geometry	61
4.18 Error Output	62
4.19 Error Output Continued	62
4.20 Load Project	63
4.21 Dipole Geometry	64
4.22 Dipole Surface Currents	64
4.23 Radiation Patterns for $\lambda/2$, λ , $5\lambda/4$, and $3\lambda/2$ Dipoles	65
4.24 Yagi Array Geometry	67
4.25 Surface Currents Due to Source at Origin	68
4.26 Radiation Pattern	68
4.27 Vertex Coordinates to Combine	71
4.28 Halfwave Dipole at Height of $\lambda/2$	72
4.29 Halfwave Dipole at Height of λ	73
4.30 Surface Currents for Dipole Height of $\lambda/2$	74

CHAPTER 1

Introduction

The intent of this project is to improve user access to software based on Moment Method (MM) and extended using multilevel fast multipole algorithm (MLFMA) developed at Oklahoma State University Robust Electromagnetic Field Testing and Simulation (REFTAS) Laboratory. Development of this software has spanned many years and the capabilities have increased as the work has progressed. At the outset, the MLFMA software was contained in a multitude of individual files with each file containing instructions to build a single geometry and then model it according to hard-coded parameters. Each geometry file runs from the Linux command line environment by calling an executable compiled from files written in C++ programming language. As a result, whenever changes needed to be made to either the model conditions, geometry build criteria, or data display methods, the source code would need to be edited using a command line editor, code revisions made, recompiled, and then linked to core routines. Execution from the command line meant that all user parameters had to either be hard-coded into the source code or typed in response to command line queries for input values. After user input was gathered, the software would produce a multitude of data printed to the command line or written to data files, the culmination of which was the displaying of the resultant re-radiated field strengths.

This software is capable of finding fields scattered due to an externally incident field or finding the radiation due to a directly applied voltage source. Despite having the processing power and numerical capabilities to solve scattering and radiation problems, it did not facilitate user interaction. In order to build a custom geometry,

an entirely new source file had to be created. In most instances, the user had no visual feedback verifying that the geometry to be modeled was what they had intended and no interim opportunity to change anything about the modeled environment. By default, model-produced standard output was printed to the command line or printed to file. All resultant data pertaining to the geometry and the induced surface currents were invisible to the user and irretrievable after completion of the program. This left the user with a very limited selection of pre-compiled options and little opportunity for adjustment of non user-controlled inputs aside from modifying the C++ code and recompiling against the core files. To compile modified code, a user would have to be sitting at a REFTAS workstation or accessing one via a VPN connection running Secure Shell (SSH). For a user to fully modify and utilize the software's capabilities would require understanding of, or familiarity with, basic command line interaction, code modification using workstation editor software, C++ programming language, and the general layout and function structure commonly used to create a geometry file. This resulted in severe limitations to the software's approachability by the average user.

The goal of this project is to improve accessibility to the average user and provide a method by which structures can be built from individual pieces including, but not limited to, wires, plates, apertures, spheres, cubes, and cavities. The main improvements to be made included a point-click interface for building shapes to be modeled, a method for viewing and modifying geometry elements, access to the most commonly used modeling parameters, a method for visualization of induced surface currents and re-radiated field magnitudes, and the ability to walk away from the terminal while it processed data without concern for the resultant command-line output being lost to inadvertent log-off or restart. The primary project goals are listed in Table 1.1

Goal No.	Description
1	Facilitate Creation of Custom Geometries
2	Increase Approachability of Modeling Capabilities
3	Reduce Learning Curve Required for Proficiency
4	Expand Data Storage and File Retrieval Options
5	Improve Data Presentation
6	Remove Access Limitations

Table 1.1: Primary Thesis Goals

CHAPTER 2

Basic Computational Electromagnetic Code Theory

Determination of re-radiated fields from complex geometries in the presence of an incident electric field is of particular importance in the field of antennas and propagation in that it gives insight into the radiation characteristics of the geometry under examination. The law of reciprocity states that, for a given antenna structure, the radiation pattern of the antenna during reception of incident electric fields is the same as that of the antenna during transmission [1]. Therefore, calculation of a given geometry's re-radiated pattern simultaneously yields the radiation pattern that would result from its use as a transmitter.

2.1 Incident Fields and Induced Currents

In the presence of an incident electric field, surface currents are induced on the surface of a geometry. These surface currents take the form of conducting currents on an ideal conductor or equivalent currents on a dielectric interface. These induced surface currents re-radiate scattered fields that, in turn, induce new surface currents. The total field is ultimately found by adding the incident source field to the re-radiated scattered fields to be solved.

2.2 Geometry Discretization

For any sufficiently complex geometry, the surface currents cannot be solved in closed form and instead must be solved numerically. By dividing (or discretizing) the geometry into N surfaces (or elements) over which the current can be represented by

basis functions, the field contribution from all elements can be used to find the surface current on any one element. Simultaneous performance of this process effectively discretizes the integral equation and produces a linear system of N equations and N unknowns.

The Moment Method (MM) uses basis functions to model the surface current on each element, each possessing an unknown weighting that must be solved. The linear system of N equations and N unknowns is then achieved by adding the scattered field created by the N basis functions to the incident source field and matching the boundary conditions at each of the N surfaces. Surface current modeling using basis functions is covered in C. W. Steele [2].

2.3 Iterative Solvers and Preconditioners

Smaller systems formed from the Moment Method can be solved by direct solution using LU factorization. However, larger systems necessitate acceleration of the MM system solution by use of MLFMA. This is due to the fact that, when large numbers of unknowns exist, LU factorization is unfeasible and iterative methods are employed to solve the linear system. Some commonly used iterative methods include the generalized minimal residual method (GMRES), biconjugate gradient method (BiCG), and biconjugate gradient stabilized method (BiCGSTAB), to name a few [3].

In matrix notation, the problem is written as $\bar{V}_m = \bar{\bar{Z}}_{mn} * \bar{I}_n$ where \bar{V}_m is the field voltage, $\bar{\bar{Z}}_{mn}$ is the interaction matrix containing the basis function representation of the each element, and \bar{I}_n are the coefficients used to represent the constant surface current magnitude on each element. Solving for the coefficients results in $\bar{I}_n = \bar{\bar{Z}}_{mn}^{-1} * \bar{V}_m$ where $\bar{\bar{Z}}_{mn}^{-1}$ is referred to as the inversion matrix.

Interaction matrices resulting from the moment method matrix are often poorly conditioned and iterative techniques converge slowly. Preconditioning can be used to improve the convergence of the interaction matrix. The preconditioned system

$\overline{\overline{M}}^{-1} * \overline{\overline{Z}} * \overline{I} = \overline{\overline{M}}^{-1} * \overline{V}$ is solved, where $\overline{\overline{M}}^{-1}$ is the preconditioning matrix and approximates $\overline{\overline{Z}}^{-1}$. Some of the most common preconditioners are Jacobi, incomplete LU factorization (ILU), and sparse approximate inverse (SAI) [4]. Preconditioners and the trade-offs of their implementation are discussed in Barrett [5].

2.4 Calculate Currents and Fields

Direct factorization can now be used to solve smaller problems or problems where moment method alone is employed. For larger problems utilizing MLFMA, preconditioning of the sparse near-field matrix can be used to accelerate the convergence of iterative methods. Convergence is achieved when the last iteration brings the error magnitude below a pre-defined threshold. Once the error is reduced to an acceptable value the solution is complete. The solved interaction matrix represents the physics of the interactions between the basis currents on every element, and can be used to find the fields present anywhere in the near-field or the far-field.

In backscattering problems, the source and observation angles are collocated as they rotate around the geometry. Therefore the system must be re-solved at each source/observation angle. In these problems, modification of the source results in a new excitation vector \overline{V} . However, the interaction matrix $\overline{\overline{Z}}$ iteratively solved above remains valid for any source and does not require recalculation.

2.5 EM Software

The procedure described above is typically carried out by computational EM software in the following steps:

1. Discretize subject geometry into individual surface patches and wire segments.
2. Create moment-method basis function set to represent the surface currents present on the resultant patches and segments.

3. Find the interactions between the surface current basis functions and the field testing functions.
4. Generate the field source due to an incident plane wave, a voltage across a patch or a segment on the geometry, or an infinitesimal (Hertzian) dipole in space.
5. Solve the resultant linear system of N equations and N unknowns by using direct matrix factorization (MM) or an iterative solution (MM or MLFMA) to get currents.
6. Determine the scattered/radiated field from the currents.

Geometry discretization in step 1 is accomplished by the Geometry Builder GUI (Section 3.3.3) and data pertaining to the resultant geometry is stored in a “geo” interface geometry file (Section 3.2.1). Basis function creation in step 2 and interaction determination in steps 3 are both accomplished within the “scatter” and “radiation” modeling executables ¹ called by the System Modeler GUI (Section 3.3.8); no output files are produced by either. The field sources required for step 4 are contained in an “src” field source file (Section 3.2.3) and are passed to the modeling executables by the System Modeler GUI. Solution of the linear system in step 5 and determination of fields from the surface currents in step 6 also take place within the modeling executables. Surface current data and scattered/radiated field data are retrieved by the System Modeler GUI and stored as surface current files (Section 3.2.4) and field magnitude files (Section 3.2.5) for later visualization by the surface current viewer (Section 3.3.9) and field magnitude viewer (Section 3.3.9).

¹Function calls to the computational EM software developed at REFTAS Laboratory are made from within the modeling executables

CHAPTER 3

File Structure and Functional Blocks

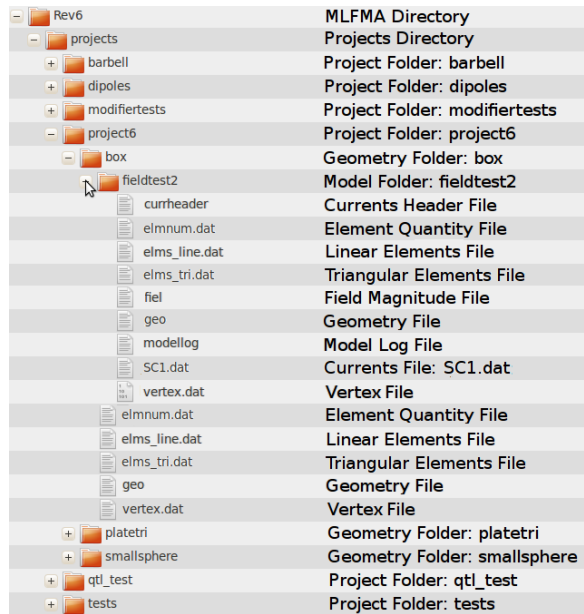
Several factors were taken into consideration when selecting the platform for the GUI. Due to the massive computational workload taking place, it made sense to store and retrieve data as often as possible to minimize the memory and processing requirements of the software. Therefore, any platform selected would need to offer the flexibility of being able to read from and write to multiple file types. It would also require the ability to obtain detailed information about the file system to facilitate successful absolute referencing of files. The computational code does not provide visual feedback to the user and it made sense that the user would want access to this information to see what is happening. Therefore, the selected platform would need to offer multiple methods of visualizing and graphically interacting with the data. Additionally, the ability to retrieve and reuse the data outside the interface would allow users to re-utilize the data without the need to rebuild the model, a process that can be extremely time intensive. Therefore, any platform chosen would need to have multiple entrance and exit points to preclude the requirement that all commands be run end-to-end. Furthermore, in order to provide access to all programming options, both current and yet to be produced, would require the ability to accommodate a large library of commands and thousands of lines of control code without getting bogged down. The platform would need to offer modular development options to provide flexibility in provision of future visual and functional updates. Lastly, the platform would need to be prevalent in both the academic and professional world to prevent the tethering present in the software's current form.

Mathworks, Incorporated's MATLAB language and computing environment [6] meets all requirements. It has widespread use in both academia and the commercial world and most potential users would already be familiar with it. Data visualization is fast and efficient and multiple options are available for reading from and writing to both text and data files. MATLAB GUI development provides an effective interface with a multitude of instructional videos, forum discussions, and code examples publicly available. Also, MATLAB served as the platform for a forerunner effort by student Ryan Salisbury [7]. His project standardized the format for interface geometry files (see Section 3.2.1) and established the procedure for obtaining surface currents from within the modeling executable. Lastly, MATLAB allows executables to be run from the command line, a necessary feature when MLFMA functions are contained in command line executable files. Looking ahead, MATLAB also has the ability to compile a GUI and its associated files into an executable that can be run independent of a MATLAB installation.

3.1 Directory Hierarchy

To understand the inner workings of the GUI, it is necessary to discuss each functional block separately (see Section 3.3). Before going into the details of each functional block, it is important to understand how data is stored and retrieved (see Section 3.2). And before discussing how data is handled, some explanation of the directory hierarchy needs to be given. The folder structure is responsible for separating and maintaining all operational, geometry, model, and log files. Each of the four folder types has a unique purpose and holds specific files required by the GUI. Figure 3.1 shows a typical GUI file hierarchy and identifies what each file and folder represent.

Each directory type and file type are detailed in the sections that follow. To provide the maximum amount of operational flexibility the GUI will operate regardless of what names are selected for the MLFMA, project, geometry, and model directories. In fact, changing the directory names between building /modeling sessions is expected and encouraged. As more and more files are produced, users will develop their own personal naming scheme that helps them



Rev6	MLFMA Directory
projects	Projects Directory
barbell	Project Folder: barbell
dipoles	Project Folder: dipoles
modifiertests	Project Folder: modifiertests
project6	Project Folder: project6
box	Geometry Folder: box
fieldtest2	Model Folder: fieldtest2
currheader	Currents Header File
elnum.dat	Element Quantity File
elms_line.dat	Linear Elements File
elms_tri.dat	Triangular Elements File
fiel	Field Magnitude File
geo	Geometry File
modellog	Model Log File
SC1.dat	Currents File: SC1.dat
vertex.dat	Vertex File
elnum.dat	Element Quantity File
elms_line.dat	Linear Elements File
elms_tri.dat	Triangular Elements File
geo	Geometry File
vertex.dat	Vertex File
platetri	Geometry Folder: platetri
smallsphere	Geometry Folder: smallsphere
qtl_test	Project Folder: qtl_test
tests	Project Folder: tests

Figure 3.1: Typical Directory Hierarchy

logically sort through folders. The only caveat to this liberty is that the MLFMA directory must have a subfolder named “projects” at the time of invocation or one will automatically be created. The GUI will only accept project pathways to absolute folder references terminated in a “projects” directory. Anything else will produce an error.

3.1.1 MLFMA Directory

All files necessary for the correct operation of the MLFMA GUI are located in the MLFMA directory. These files include the geometry executables necessary for generation of geometry elements, the MLFMA core code responsible for performing the modeling, MATLAB conversion codes used to generate geometry interface files, and the actual GUI figure layout and control code. To maximize operational flexibility, the GUI can be called from the MLFMA directory regardless of where that directory is absolutely stored within the file system. This is accomplished by creating an absolute reference to the MLFMA directory at the time of invocation. This feature allows the GUI files to be run from a thumb drive, back-up hard drive, MATLAB directory, or any other file storage location. The only limitation to hosting options is the requirement that users have the ability to write to the MLFMA directory and any nested subdirectories. File types commonly encountered and their functions are provided in Table 3.1.

File Name	File Type	Description
buildhelix.m	MATLAB Function	Helical geometry builder
builder	C++ Executable	Basic geometry builder
cline.m	MATLAB Function	Plots segments using colorbar information [8]
iconvertall.m	MATLAB Function	Converts original surface current files to SC*.dat files
geoconvert.m	MATLAB Function	Converts Interface Geometry Files to Delaunay Geometry Files
georevert.m	MATLAB Function	Rebuilds Interface Geometry Files from Delaunay Geometry Files
mlfma.fig	MATLAB GUI Layout	Layout of project loader GUI
radiate	C++ Executable	Radiation modeler
scatter	C++ Executable	Scattering modeler
quadmesh.m	MATLAB Function	Plots quadrilateral elements using Delaunay format [9]

Table 3.1: Files Common to MLFMA Directory

3.1.2 Projects Directories

The only required sub-folder within the MLFMA directory is the “projects” directory. This folder stores projects separately from each other and within which all geometry creation and subsequent modeling takes place. Creating a new project in the GUI results in the creation of a new project folder within this directory. Storing each project in its own project folder provides logical sorting of user-generated geometries into projects identified by their respective end product. To maximize user access, the overarching projects folder can be stored in multiple locations for access by different users. The only limitation is that each folder must be named “projects” to be acceptable to the GUI. The default projects folder location is within the MLFMA directory. If no projects sub-folder exists at the time of invocation, one will be created automatically. The names of the files present in the default projects folder are used to populate the list of existing projects available for selection in the project loader covered in Section 3.3.1.

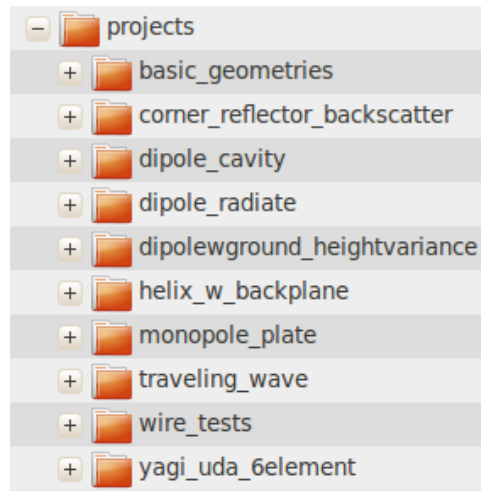


Figure 3.2: Projects Folder Contents

3.1.3 Geometry Directories

Within each project directory are geometry directories associated with that project. Each time a new geometry is built, a new geometry folder is created to house the files that define it. Each geometry folder is named according to the shape it represents, as defined by the user at the time of creation, and all geometry folders contain the same filenames. Geometry folders can be produced as a result of the Build function block (Section 3.3.3), user-created MATLAB functions, or the copying and combination of other geometries. Every geometry directory holds three distinct file types: interface geometry files produced by the builder and passed to the modeler, log files that store creation and modification information, and Delaunay geometry files that are used for visualization and modification of the geometry data. These file types and their descriptions are discussed in Section 3.2. After creation, these files are read, modified, and re-written by the other functional blocks, and serve as the basis for system modeling. Modeling cannot be performed without first identifying the geometry folder containing the element file to be modeled.

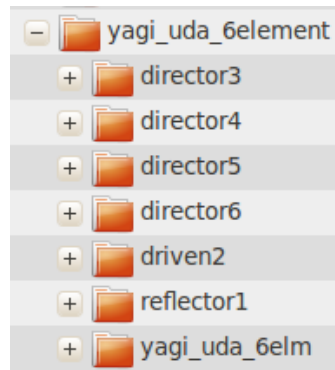


Figure 3.3: Geometry Directories

3.1.4 Model Directories

Whenever a geometry is modeled, a new model folder is created to house the resultant output files in order to keep files associated with a particular model from being incorrectly associated with others. Sorting the model data in this manner serves to logically sort the hundreds of potential output files and allows the user to retrieve all files attributed to a particular model without having to re-model the system. This separation also allows the same naming scheme to be used within every model folder. Each model folder contains six file types responsible for storing surface current and field data, model log files, and the original geometry data being modeled. Copying the geometry files allows accurate interpretation of the model data in relation to the geometry data even if the underlying data in the geometry folder is modified. This effectively produces a snapshot of the geometry at the time of modeling that will remain unmodified even if the geometry itself is modified.

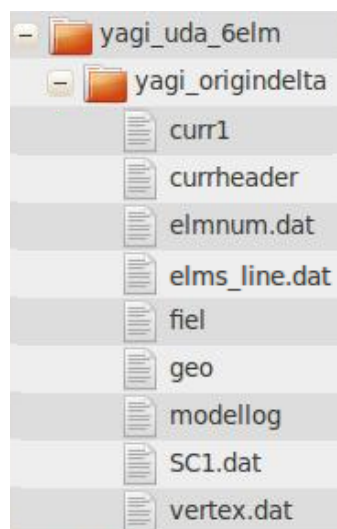


Figure 3.4: Model Directory

3.2 File Types

The MLFMA GUI utilizes six different types of data files to store geometry and model information. Each file type retains its standardized name regardless of the folder in which it is located. This naming scheme facilitates location of files and prevents the need for a file map to be built. The different type of data files, their names, and a description of each file are provided in Table 3.2.

File Type	File Names	Description
Interface Geometry Files	geo	Geometry Format Expected by Modeler
Delaunay Geometry Files	vertex.dat	Vertices as Cartesian Coordinate Triplets
	elms.line.dat	Line Elements Comprised of vertex.dat Row Indices
	elms.tri.dat	Triangular Elements Comprised of vertex.dat Row Indices
	elms.quad.dat	Quadrilateral Elements Comprised of vertex.dat Row Indices
	elnum.dat	Quantity of Each Element Type
Field Source Files	src	Source(s) to be Modeled
Surface Current Files	curr#	Surface Current Vectors per Element
	SC#.dat	Average Normalized Surface Current Magnitude per Vertex
	currheader	Surface Current Header File
Field Pattern Files	fiel	Field Magnitude for Given Observation Angle
Log Files	log	Log of Geometry Creation
	modellog	Log of Model Creation

Table 3.2: File Types

3.2.1 Interface Geometry Files

Interface geometry files, or “geo” files, are created by the builder function block executables and is typically generated by the C++ builder executable. They are referred to as interface files because their sole responsibility is to interface with the modeler executables, supplying easily assimilated element information upon which the model is based. Every geo file follows a strict format that allows multiple element types to be contained within the same file. Each row represents a unique element, the type of which is designated by the first character of each row. An “L” represents a line, “T” represents a triangle, and “Q” represents a quadrilateral. Vertex information then follows the alpha character. Line elements are nine characters long and contain the alpha designator followed by the Cartesian coordinate triplets of two vertices, the segment radius, and the resistance per wavelength of the wire segment. Triangular elements are ten characters long and contain the alpha designator followed by the Cartesian coordinate triplets of three vertices. Quadrilateral elements are similar to triangular elements but with twelve characters representing Cartesian coordinate triplets for four vertices instead of three. Figure 3.5 shows the formatting of each element type within a geo file. The geo file is created at the time the geometry is

Q		0.8	1.4	0	0.8	1.5	0	0.9	1.5	0	0.9	1.4	0
Q		0.9	0.5	0	0.9	0.6	0	1	0.6	0	1	0.5	0
Q	Q	0.9	0.6	0	0.9	0.7	0	1	0.7	0	1	0.6	0
Q		0.9	0.7	0	0.9	0.8	0	1	0.8	0	1	0.7	0
T		1.5	1.5	2	1	1.5	2	1	1	2			
T		1.5	1.5	2	1	1	2	1.5	1	2			
T	T	1.5	1	2	1	1	2	1	0.5	2			
T		1.5	1	2	1	0.5	2	1.5	0.5	2			
T		1	1	2	0.5	1	2	0.5	0.5	2			
L		0	0		1.37255	0	0	1.47059	0.001	0			
L		0	0		1.47059	0	0	1.56863	0.001	0			
L		0	0		1.56863	0	0	1.66667	0.001	0			
L	L	0	0		1.66667	0	0	1.76471	0.001	0			
L		0	0		1.76471	0	0	1.86275	0.001	0			
L		0	0		1.96078	0	0	2.05882	0.001	0			

Figure 3.5: Interface Geometry File Format

created. It is never modified or edited, but it can be overwritten with updated geometry information via the georevert.m MATLAB function. The georevert.m MATLAB

function rebuilds the interface geometry file from the appropriate Delaunay geometry files. Quadrilateral elements are converted first, followed by triangular elements, and finally linear segments. Although interface files are not required to follow elemental grouping or ordering, a geo file produced by the `georevert.m` function exhibits both elemental grouping and elemental ordering and follows the same strict formatting of the original interface geometry files. Prior to modeling, the geo file is rebuilt using the `georevert.m` function to capture any modifications made to the Delaunay geometry files by the updater function block discussed in Section 3.3.5. This step is necessary to standardize the element order prior to interfacing with the modeler executables. Once passed to the modeler executable, the C++ source code unpacks each row of the geo file into its corresponding element within the elements array. More information on the modeler function is discussed in Section 3.3.8.

3.2.2 Delaunay Geometry Files

Delaunay geometry files contain the same information as interface geometry files, but in Delaunay format. Delaunay format stores all the vertices in a single file and stores vertex relationships in separate element files. The element files contain all the same relationship information as the interface geometry files, but require indexing into the vertex file to obtain the actual Cartesian coordinate triplet of each vertex. The resultant vertex file contains three columns representing point values in X-, Y-, and Z-coordinates, effectively representing one vertex per row. The three element files (`elms_line.dat`, `elms_tri.dat`, and `elms_quad.dat`) represent linear, triangular, and quadrilateral elements and contain two, three, or four vertex values per row. These integers reference the rows in the `vertex.dat` file that contain the Cartesian coordinate triplets of which each element is composed. Figure 3.6 shows the formatting of each type of .dat file. After modeling is completed, the `geoconvert.m` function is again called to produce vertex and element files. This guarantees accurate surface current

visualization by insuring the vertices are retrieved from the modeler in the same order that they were sent. Because the underlying MATLAB geometry files associated with the parent directory can be changed by the modifier at any time, especially after subject model is produced, these files must be generated and segregated at the time of modeling. Otherwise, the user runs the risk of using modified .dat files that are invalid for representation of the modeled system. Since the geo file is never modified, the Delaunay geometry files are directly updated by the updater function block and ultimately serve to rebuild the interface geometry files when it is time to model the system or save permanent changes to the geometry.

X	Y	Z	V1	V2	R	RPL	V1	V2	V3	V4	V1	V2	V3
1.1	1.2	0	30	31	0.001	0	59	61	75	74	198	193	192
1.1	1.3	0	31	32	0.001	0	61	62	76	75	198	192	197
1.1	1.4	0	32	33	0.001	0	62	63	77	76	197	192	186
1.1	1.5	0	33	34	0.001	0	63	64	78	77	197	186	196
1.2	0.5	0	34	35	0.001	0	64	65	79	78	196	186	180
1.2	0.6	0	35	36	0.001	0	65	67	80	79	196	180	195
1.2	0.7	0	36	37	0.001	0	67	68	81	80	195	180	178
1.2	0.8	0	37	38	0.001	0	68	69	82	81	195	178	194
1.2	0.9	0	38	39	0.001	0	69	70	83	82	193	133	132
1.2	1	0	39	40	0.001	0	70	71	84	83	193	132	192
1.2	1.1	0	40	41	0.001	0	74	75	86	85	192	132	126
1.2	1.2	0	41	42	0.001	0	75	76	87	86	192	126	186
1.2	1.3	0	24	25	0.001	0	76	77	88	87	186	126	120
1.2	1.4	0	25	26	0.001	0	77	78	89	88	186	120	180
1.2	1.5	0	26	27	0.001	0	78	79	90	89	180	120	118
1.3	0.5	0	27	28	0.001	0	79	80	91	90	180	118	178
1.3	0.6	0	28	29	0.001	0	80	81	92	91	133	73	72
1.3	0.7	0	29	30	0.001	0	81	82	93	92	133	72	132
1.3	0.8	0	42	43	0.001	0	82	83	94	93	132	72	66
1.3	0.9	0	43	44	0.001	0	83	84	95	94	132	66	126
1.3	1	0	44	45	0.001	0	85	86	97	96	126	66	60

Figure 3.6: Delaunay Geometry File Format

3.2.3 Field Source Files

Field source files store field source information in the form of an “src” file. Figure 3.7 shows the contents of a typical “src” file. When the “radiate” system modeler executable is called, the source information is unpacked from within the

modeler executable and each source represented individually as either an incident plane wave, a voltage source

P	45	45	1	0	NaN	NaN
D	0	0	1	1	0	NaN
D	0	0	-1	0	-1	NaN
H	1	0	0	1	0	0
H	-1	0	0	0	1	0

Figure 3.7: Field Source File

on a geometry patch or segment, or as an infinitesimal (Hertzian) dipole in space. Within the “src” file, each source is stored in its own row and designated by a single alpha character. Plane waves are designated by a “P” and have four characteristic values which define the incidence angle and plane wave polarization. Delta sources (voltages on patches or segments of the geometry) are designated by a “D” and have five characteristic values which define the Cartesian coordinate location and complex voltage. Hertzian dipole sources are designated by an “H” and have six characteristic values which define the Cartesian coordinate location and dipole orientation vector in Cartesian unit-vectors. To facilitate manipulation of the “src” file from within the radiation system modeler, not-a-number (NaN) values are used to fill space and make each row seven entries long. The radiation modeler GUIs responsible for addition of sources to the “src” file are shown in Figure 3.8. Sources are loaded into the system

Figure 3.8: Add Field Source GUIs

model one at a time and are, by default, additive in nature. More information on the radiation system modeler is discussed in Section 3.3.8.

3.2.4 Surface Current Data Files

Surface current files are generated by the modeler executables and contain both the surface current magnitude on each vertex and the surface current vectors on each element. A “currheader” file is also created and populated with one row of data for each current file produced. Each row identifies the observation angle elevation and azimuth (θ and ϕ) as well as the

model type: scattering or radiation. The currheader file serves to populate the current-viewer dropdown menu when an appropriate geometry is loaded to the workspace. Figure 3.9 shows the formatting of both surface cur-

curr1									
Theta = 45 Phi = 45									
-0.966667	-1.43333	0	1.08501e-05	3.29154e-05	0	0.005	0.00693151		
-0.933333	-1.46667	0	1.71934e-05	1.20144e-05	0	0.005	0.00419504		
-0.966667	-1.33333	0	5.77916e-06	5.01492e-05	0	0.005	0.0100962		
-0.933333	-1.36667	0	9.72389e-06	4.59007e-05	0	0.005	0.00938308		

C65.dat			
-1	-1.5	0	0.0055633
-1	-1.4	0	0.0088039
-1	-1.3	0	0.010765
-1	-1.2	0	0.010278
-1	-1.1	0	0.0082361
-1	-1	0	0.0057015

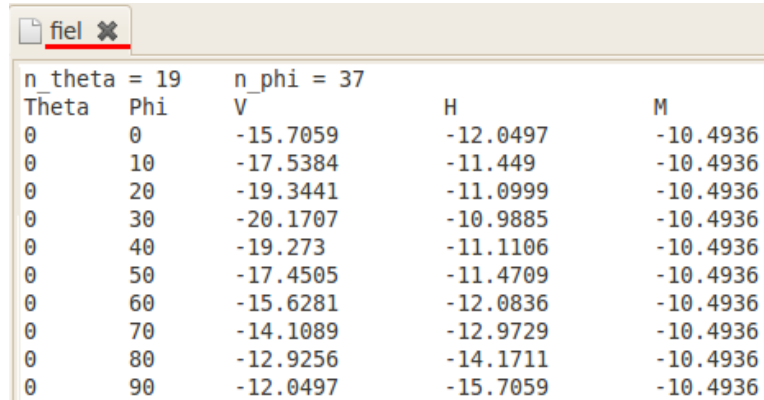
currheader			
1	Theta = 45	Phi = 45	[Bistatic Backscatter]

Figure 3.9: Surface Current Data File Format

rent data file types and the currheader file. Radiation and bistatic scatter modeling produce one surface current file that is valid for all incident angles. Monostatic scatter modeling, or backscatter modeling, produces surface current files for every scattering angle. Files produced by the model executables follow the naming scheme curr1, curr2, and so on. These files contain the surface current vectors on each element. After modeling is complete the iconvertall.m function is called. This function calculates the surface current vector magnitude normalized by the hypervolume for each element, identifies all elements touching each vertex, finds the average normalized surface current magnitude of all connected elements, and assigns that value to the vertex. This data is appended to vertex.dat and re-saved as SC1.dat, SC2.dat, etc. This conversion reduces disk space usage by up to 75% and greatly accelerates surface current visualization as the resultant format can be plotted directly.

3.2.5 Field Intensity Files

Field intensity files store the radiation intensity measured in the far-field as orthogonal plane-wave components E_{θ_scat} and E_{ϕ_scat} . These components represent the complex intensity in the θ and ϕ vectoral directions, respectively. Observation values in θ and ϕ are followed by three magnitudes. These three magnitudes represented at each incident angle represent the vertically polarized component of the field, the horizontally polarized component of the field, and the magnitude of both the vertically polarized and horizontally polarized field components. Formatting of the fiel file is show in Figure 3.10. The fiel file is directly responsible for creation of the View Field plots. Each of these plots can be set to display the appropriate column representing either the vertical component, horizontal component, or component magnitude when selected in the corresponding dropdown menu.



The screenshot shows a file editor window with the title 'fiel'. The window contains a table with 5 columns: 'Theta', 'Phi', 'V', 'H', and 'M'. The first two columns, 'Theta' and 'Phi', have a range of values from 0 to 90 in increments of 10. The remaining three columns, 'V', 'H', and 'M', contain numerical values. The table is as follows:

Theta	Phi	V	H	M
0	0	-15.7059	-12.0497	-10.4936
0	10	-17.5384	-11.449	-10.4936
0	20	-19.3441	-11.0999	-10.4936
0	30	-20.1707	-10.9885	-10.4936
0	40	-19.273	-11.1106	-10.4936
0	50	-17.4505	-11.4709	-10.4936
0	60	-15.6281	-12.0836	-10.4936
0	70	-14.1089	-12.9729	-10.4936
0	80	-12.9256	-14.1711	-10.4936
0	90	-12.0497	-15.7059	-10.4936

Figure 3.10: Backscatter Field Intensity File Format

3.2.6 Log Files

Log files are created whenever the builder or modeler executables are called to document the settings used to generate the resultant geometry of model files. Generation of geometry data produces a “log” file and generation of model data produces a “modellog” file. The log file contains all the information sent to and generated by the geometry builder executable. It serves as a birth certificate by recording the timestamp for the associated part and contains all the information that would be needed to rebuild it exactly. It also displays how many vertices and elements were created by the `geoconvert.m` function. Whenever a geometry is modified, the log file is appended to include the steps that were carried out and the time and date that the modifications took place. Figure 3.11 shows a log file for a newly built geometry.

```
line 0 = /media/9C8F-4A13/ECEN5070_Thesis/new_files/MLFMA_GUI/RevA/builder
line 1 = 13
line 2 = director3
line 3 = .001
line 4 = .289
line 5 = 0
line 6 = .218
line 7 = .289
line 8 = 0
line 9 = -.218
line 10 = .003369
line 11 = 0
line 12 = true
```

Figure 3.11: Log File Information

The modellog file contains all the model settings and applicable information displayed in the MATLAB command line during model creation. This is the text that the original MLFMA executables output to the command line with the only difference being that it is a verbatim log of the output data produced in the MATLAB command line environment instead of the Unix CLI. As a result, these files can be thousands of lines long when the system is iteratively solved for multiple incidence angles. Example modellog output for a modeled geometry is shown in Figure 3.12.


```

02-May-2011 16:18:46
Function Call = /media/9C8F-4A13/ECEN5070_Thesis/new_files/MLFMA_GUI/RevA/model_deltasource
Geometry File = /media/9C8F-4A13/ECEN5070_Thesis/new_files/MLFMA_GUI/RevA/projects/
yagi_uda_6element/yagi_uda_6elm/geo
EM TYPE = MLFMA [default = MLFMA]
Levels = 1 [default = 1]
Significant Digits = 3 [default = 3]
Matching = NORMAL [default = NORMAL]
Optimize for = SPEED [default = SPEED]
Corrected IBC [T/F] = false [default = false]
Quadrature Order = 6 [default = 6]
Group Size = 0.25 [default = 0.25]
SOLVER TYPE = 2 [default = GMRES]
Iterative Tolerance = .001 [default = .001]
Maximum Iterations = 280 [default = 280]
Restart = 280 [default = 280]
Trace[T/F] = false [default = false]
PRECON_TYPE = 1 [default = ILUTP]
Drop Tolerance = .001 [default = .001]
Fill Factor= 1 [default = 1]
Interaction Distance= .3 [default = 0.3]
Pivot Factor = 1 [default = 1]
Precondition side = 1 [default = RIGHT]
Theta start:stop:steps = 0:180:181 [default = 0:0:1]
Phi start:stop:steps = 0:360:361 [default = 0:180:5]
Source Location = 0:0:0 [default = 0:0:1]
droptol, fill_factor, precond_dist 0.001 1 0.3
Total nodes = 2654
2654 unknowns.
Setting up level 1 groups.
M[0] = 12
L[0] = 5
K[0] = 50
Level 1 interactions = 64 out of 144 possible.
near_interactions = 3917916
2654 unknowns.
3917916 sparse matrix elements out of a total of 7.04372e+06 total elements.
55.6229%
Setting up sparse matrix. (Multiply following by 100,000 to get number found.)
Buffers allocated
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
27 28 29 30 31 32 33 34 35 36 37 38 39 Sparse fill time = 37.69
reordering CORR_mat
sort M_div_rot
sort M_rot_div
sort M_h
Sparse fill time = 38.15

Done setting up sparse matrix, now converting.
Finalize int mat
Done converting sparse matrix.
Time = 0.06

Min diag = (0.0492044,-109.566)
Min diag number = 470
Max diag = (0.0494556,-110.548)
Max diag number = 363
Setting up translation matrices

```

Figure 3.12: Model Log File Information

3.3 Functional Blocks

The MLFMA GUI is capable of performing a multitude of complex tasks that can be logically grouped into nine functional blocks. These functional blocks perform the tasks of referencing, creating, modifying, writing, reading, interpreting, adding to, removing from, updating, and modeling the data files. The MLFMA GUI is actually comprised of four separate GUIs: the Project Loader GUI, Main GUI, Geometry Builder GUI, and System Modeler GUI. The Workspace, Modifier, Remover, Joiner, Updater, Data Visualization function blocks are all stored within the main GUI. Project Selection, New Geometry Builder, and System Modeler are all contained in their own, individual GUIs.

Function Block	GUI	Description
Project Loader	Project Loader GUI	Store User-Defined Project Path, Call Main GUI
GUI Workspace	Main GUI	Maintain Pathways to Geometries of Interest
Geometry Builder	Builder GUI	Produce Interface and Delaunay Interface Files
Geometry Modifier	Main GUI	Make Modifications to Delaunay Interface Files
Geometry Updater	Main GUI	Convert Delaunay Geometry Files to Interface Geometry Files
Element Remover	Main GUI	Remove User-Specified Elements
Element Joiner	Main GUI	Create New Vertex and Element Entries in Delaunay Geometry Files
System Modeler	Modeler GUIs	Produce Model Files from Interface Geometry Files
Data Visualization	Main GUI	Produce Graphical Representation of Data Files

Table 3.3: Function Block Descriptions

3.3.1 Project Loader

The project loader, shown in Figure 3.13, is the first screen to greet the user after the MLFMA command is executed from the MATLAB command prompt. At invocation, the absolute pathway to the MLFMA directory is determined and stored. From the project loader, users can choose to load existing projects from the default projects folder rooted in the MLFMA directory or from any other projects folder directory in the file system. To choose an existing project from the default folder, the user selects a project from the list in the middle of the project loader GUI and then clicks the load button. This list is generated from the folders located within the default projects folder rooted in the MLFMA directory. To choose existing projects from another projects folder in the file system, the user clicks on the button labeled “..” and locates the desired folder. The list is updated with all folders contained in the selected “projects” folder and the user can then select the desired project and click the load button. To create a new project folder, the user enters the desired project name in the new project box and then clicks the “create” button. Clicking on either the “load” or “create” button stores the absolute path to the selected project folder to allow absolute referencing of sub-directories and their files for use elsewhere in the MLFMA GUI. The main GUI is then opened, the MLFMA directory path and the selected project directory path passed to it, and then the project loader GUI closes.

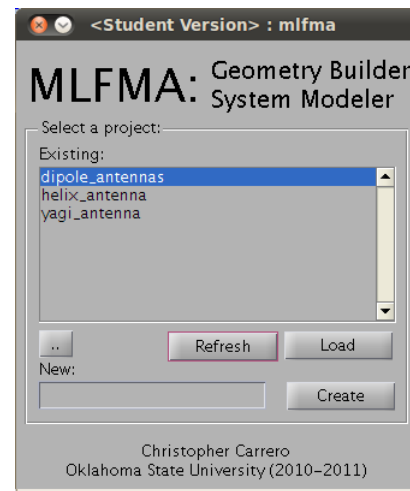


Figure 3.13: Project Loader GUI

3.3.2 GUI Workspace

The GUI Workspace component of the main GUI (Figure 3.14) is responsible for maintaining absolute references to all geometries that the user has loaded. The GUI workspace essentially serves as the function core with which all other functional blocks interact. At invocation the workspace starts out without any geometries loaded. The selected project folder is displayed in the top left corner and clicking the “Load” button allows the user to select any geometry within that project folder to load into the workspace. When a geometry folder is loaded,

it and any other geometries already present in the workspace are visualized with the Geometry Viewer. Geometries can be removed individually by selecting them and clicking on the “Unload” button or all at once by clicking on the “Reset” button. The user can select individual geometries in the workspace to be the subject of modification (Section 3.3.4), modeling (Section 3.3.5), or visualization of model-produced surface current and

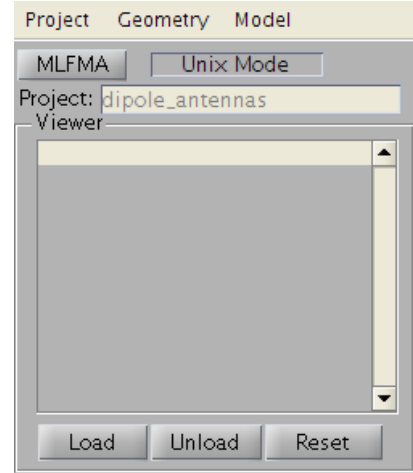


Figure 3.14: GUI Workspace

field pattern data (Section 3.3.6). At the top of the workspace is a toolbar with entries titled Project, Geometry, and Model. The Project toolbar entry allows the user to load from an existing project or create a new project within the current project folder. Either option pulls up an intermediate GUI with either a list of available projects and a “Load” button or an empty box for entry of the new project name and a “Create” button. The Geometry toolbar entry allows the user access to the geometry builder, discussed in Section 3.3.3 below. The Model entry gives the user the option to begin a backscatter, delta source, or Hertzian dipole source model. The function of these options is discussed in Section 3.3.5. Finally, clicking on the “MLFMA” button allows the user to revisit the Project Loader GUI covered in Section 3.3.1.

3.3.3 Geometry Builder

Geometry Builders are contained within their own GUI and are called when the user selects one of the options stored under the “Build” entry on the main GUI toolbar. Each Builder effectively builds a string to be executed in the Unix command line. These strings call executables based on

test_cases¹ created by Dr. West or call user-created MATLAB functions that build geometries, store them as Delaunay geometry files, and then call georevert.m to create the geo interface geometry file. At the bottom of each geometry

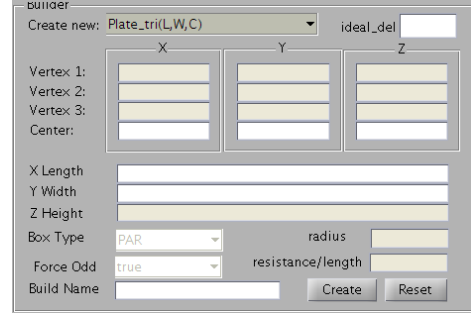


Figure 3.15: Geometry Builder

builder are two buttons. The “reset” button clears all attribute entries and resets to default values. The “build” button creates the execution string containing an absolute path to the geometry executable as well as all applicable attributes to be passed as arguments. The resultant geo and log files are moved from the MLFMA Directory to the newly created geometry directory where the geoconvert.m function creates .dat files from the geo file. The builder GUI, shown in Figure 3.15, allows the user to select from among several basic geometries, define geometry-specific attributes, and create a folder name for the resultant geometry files. The attributes available are dependent on the geometry type selected and the user’s ability to edit each input block changes according to the geometry selected. Input blocks include vertices, radius, center point, x-length, y-length, z-length, and other geometry-specific attributes (all relative to λ). Other geometries can be created using custom MATLAB functions as well, provided the resultant elements are written to Delaunay geometry files and georevert.m run. Basic geometries available through the basic builder and their input parameters are listed in Table 3.4.

¹Originally, test_case executables created a geometry, modeled it, and displayed the results in one step. Now they build the geometry and provide interface geometry files to the GUI.

Geometry	Parameters	Element Type
Plate	Length, Width, Centroid	tri or quad
	Vertex1, Vertex2, Vertex3	tri or quad
Box	Length, Width, Height, Centroid	tri or quad
Cube	Length, Width, Height, Centroid	tri or quad
Wire	Vertex1, Vertex2, Radius, Resistance/ λ	seg
Sphere	Radius, Centroid	tri
Sphere Cavity	Radius, Centroid	tri
Cube Cavity	Length, Centroid	tri or quad
Box Cavity	Length, Width, Height, Centroid	tri or quad

Table 3.4: Builder Geometries

3.3.4 Geometry Modifier

The geometry modifier is part of the main GUI and allows modification of geometries loaded to the workspace. Once a geometry is selected in the workspace, selecting an entry in the modifier will load the contents of the associated vertex.dat file, modify each row, and save over the vertex.dat file. Note that only the entries within the vertex.dat file are modified since the element files reference the vertex file by row number. These references will remain valid even after changes have been made. An important distinction to make is that only the geometry selected within the loader will be modified. To modify complex geometries as a whole the user will need to call the Combine function covered in Section 3.3.5 prior to modification. This will insure that all vertices to be modified are stored within a single vertex.dat file.

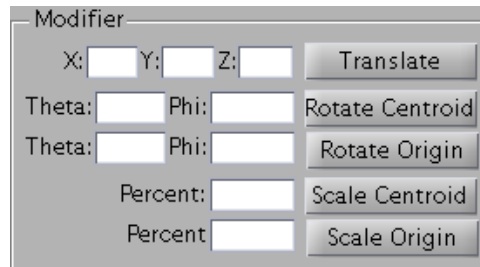


Figure 3.16: Geometry Modifier

Cartesian Translation

The “translate” command loads all vertices into an nx3 matrix of Cartesian coordinate points and then moves each vertex by the x, y, and z magnitudes defined by the user. The matrix is then re-saved to the appropriate vertex.dat file, effectively replacing every vertex in the original file with a vertex shifted by the user-defined amounts. The images below show the modifier values used and the resulting effect they have on the geometry.

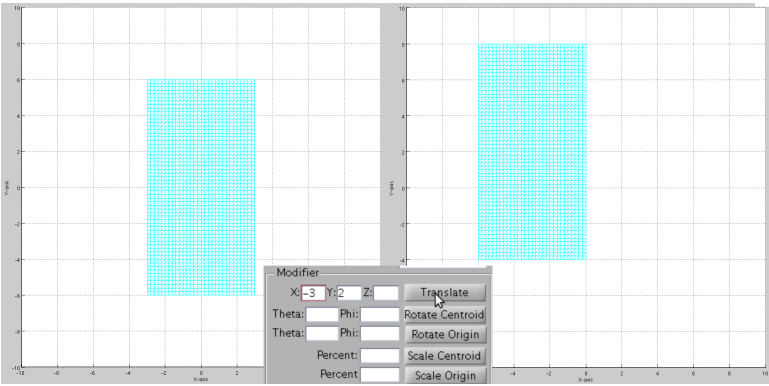


Figure 3.17: Translation in X-Y Plane

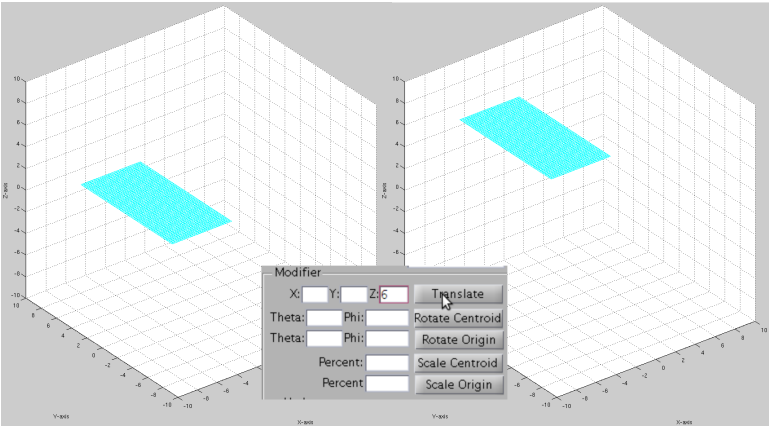


Figure 3.18: Translation in Z

Rotation with Respect to Centroid

The “rotate centroid” command loads the vertex.dat file into an nx3 matrix of Cartesian coordinate triplets and takes the average of each of the X, Y and Z columns to determine the overall centroid of the geometry. The entire array is then translated by the amounts required to shift the centroid to the origin. Each point is then converted to spherical coordinates and rotated by the user-defined θ and ϕ magnitudes. After each vertex is rotated about the centroid, the matrix is converted back to 3D Cartesian coordinates and then shifted back by the amounts required to shift the centroid back to its original position. The result is that the geometry appears to rotate about its center. Figures 3.19 and 3.20 demonstrate a flat plate being shifted about its centroid.

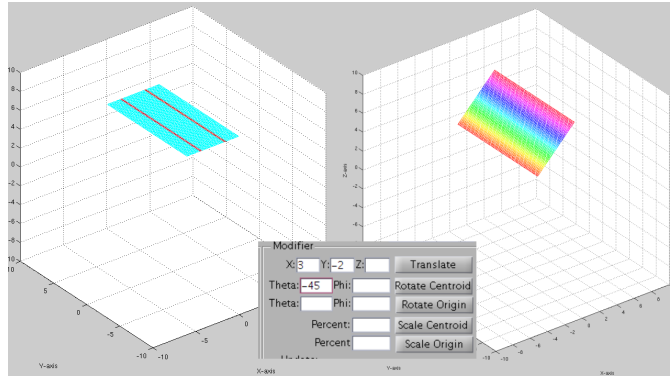


Figure 3.19: Centroid Rotation in θ

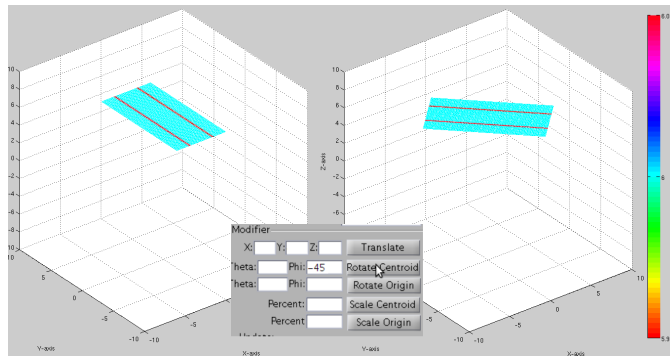


Figure 3.20: Centroid Rotation in Φ

Rotation with Respect to Origin

The “rotate origin” command loads the vertex.dat file to the workspace as an nx3 matrix and then converts the vertex array to spherical coordinates. Each vertex is then rotated by the user-defined θ and ϕ magnitudes, the array converted back to 3D Cartesian coordinates, and then re-saved as vertex.dat. As a result the geometry rotates around the origin in an orbital manner. Figures 3.21 and 3.22 demonstrate the rotation of a flat plate about the origin.

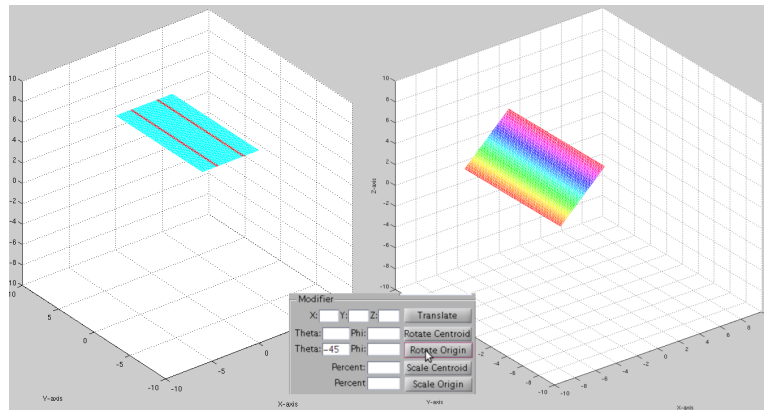


Figure 3.21: Rotate Around Origin in θ

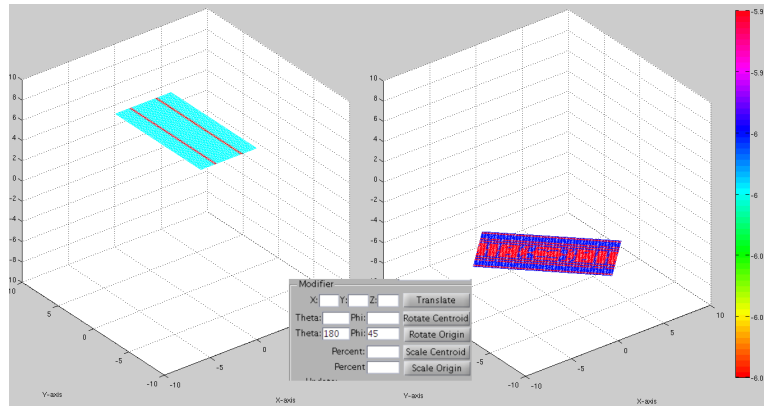


Figure 3.22: Rotate Around Origin in θ then ϕ

Scale with Respect to Centroid

The “scale centroid” command loads the contents of vertex.dat to an nx3 matrix, centroid shifted to the origin, and converts the vertices to spherical coordinates. θ and ϕ are held constant while the radius is multiplied by the percentage provided by the user. The vertices are then converted back to Cartesian coordinates, shifted back to the original centroid, and re-saved as vertex.dat. This causes the geometry to shrink or grow in place without shifting otherwise. Scaling is controlled by a percent input. Figures 3.23 demonstrates a sphere being scaled 200% with respect to its centroid.

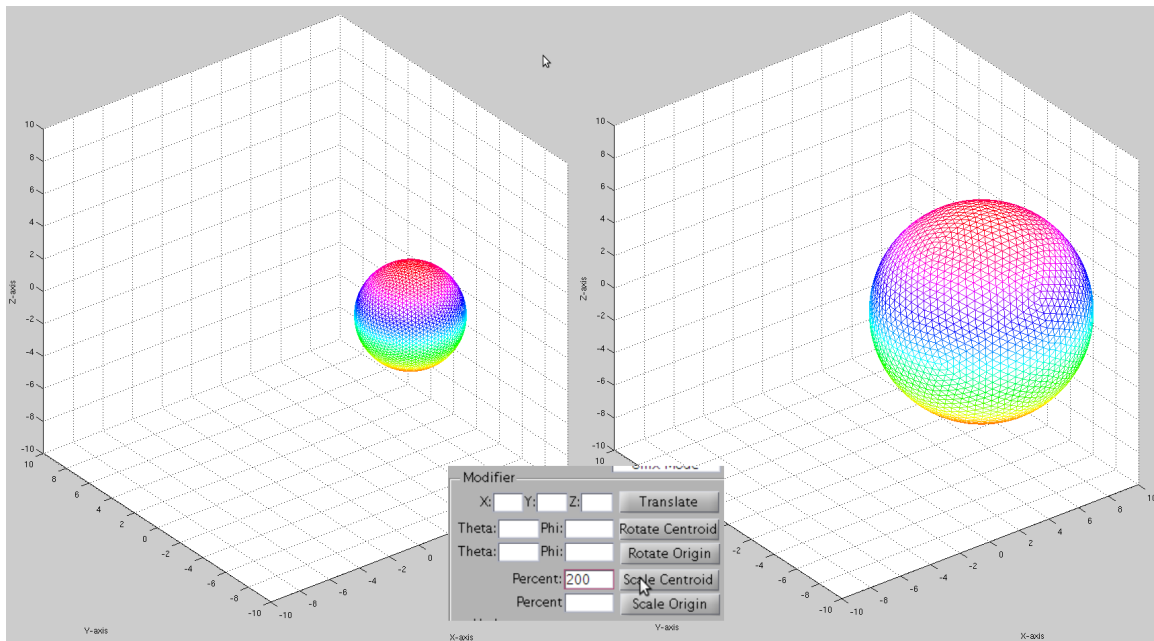


Figure 3.23: Scale Centroid

Scale with Respect to Origin

The “scale origin” command loads vertex.dat into an nx3 array and immediately converts each vertex from Cartesian to spherical coordinates. θ and ϕ are held constant while the radius is multiplied by the user-specified percentage. The vertex array is then converted back to Cartesian coordinates and re-saved as vertex.dat. This causes the geometry to shrink or grow as well as move nearer to or further away from the origin. Scaling is controlled by a percent input. Figure 3.24 shows a sphere scaled to 25% of its original size with respect to the origin. As a result the sphere is 1/4 its original size and the distance from the geometry to the origin is 1/4 the original distance.

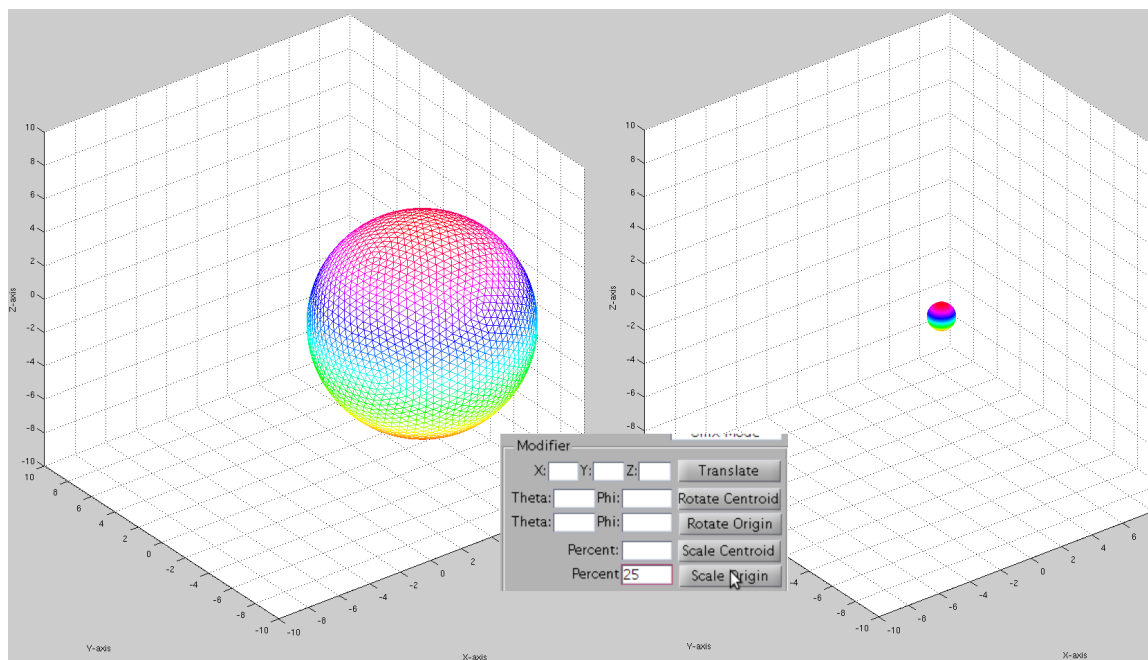


Figure 3.24: Scale Origin

3.3.5 Geometry Updater

As files undergo changes, the need arises for the user to both save their progress as well as recover from any mistakes made along the way. The updater block gives the user the ability to save their progress by overwriting the original geo file with new data as well as the ability to undo changes by reverting back to the most recent geo file configuration. Additionally, the Update block allows currently existing geometries to be copied to new geometry names and allows all geometries present in the workspace to be combined into a single geometry file. The Updater block of the GUI is shown in Figure 3.25.

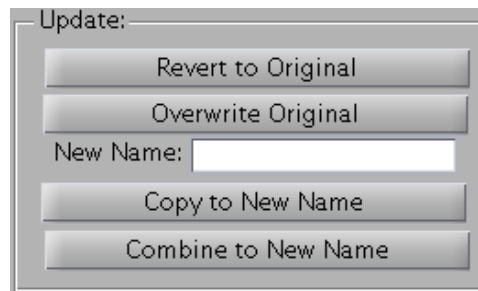


Figure 3.25: Geometry Updater

Update: Revert to Original (Undo)

“Revert to Original” allows the user to return the geometry vertex.dat file back to the state it was in at creation. This is accomplished by calling the geoconvert.m function that rebuilds the Delaunay geometry files from the original interface geometry files.

Update: Overwrite Original (Save)

“Overwrite Original” allows the user to permanently save all the changes that they have made to a geometry via modification since the time that it was created. This is accomplished by calling the georevert.m function that rebuilds interface geometry file geo from the Delaunay geometry files. After geo is rebuilt, geoconvert.m is called to rebuild the vertex.dat and elms*.dat files. This insures that, even if an overwritten

geo file is used as the basis for a model, the surface current files produced will have the same vertex order as those passed to the modeler.

Update: Copy to New Name

“Copy to New Name” gives the user the option to copy the geometry folder contents to a new folder. Therefore, if a geometry takes 10 steps to complete, the resultant Delaunay geometry files can be used to update the geo file via overwrite and can then be copied as many times as the user pleases. The copied file contains the same files as the original, and thus the same Delaunay geometry files. Therefore, when it is loaded it is indistinguishable from the original when viewed in the Geometry Viewer and can be highlighted and modified as a separate geometry from within the workspace. Figure 3.26 shows a single plate copied and translated 3 times to produce a column of duplicate plates.

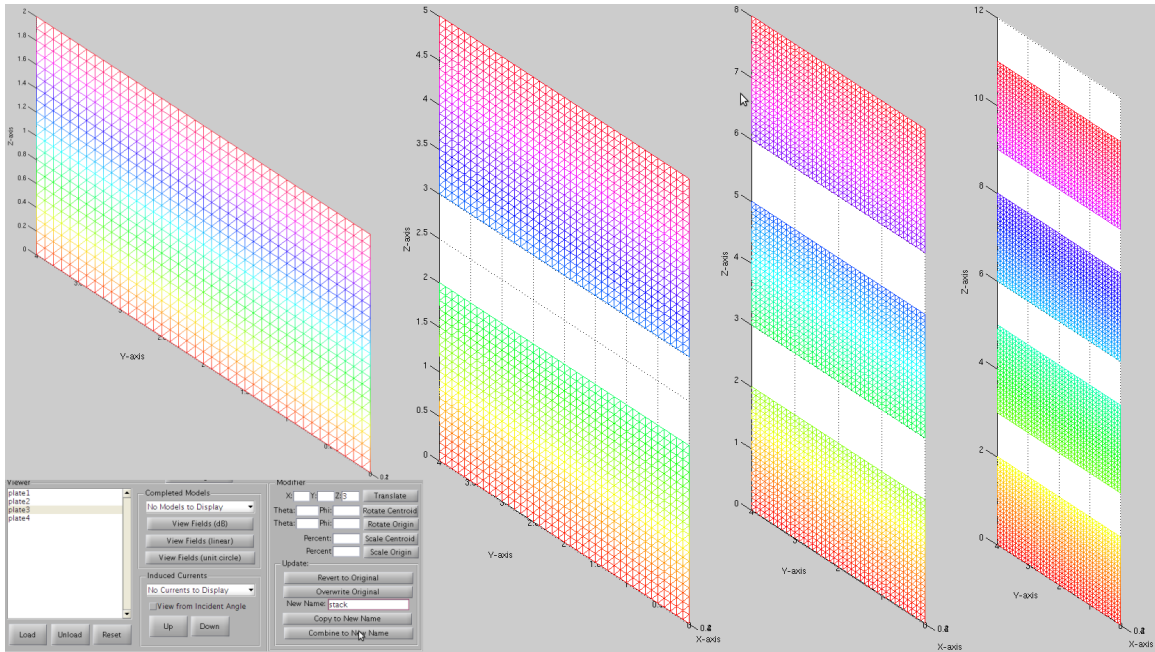


Figure 3.26: Original Plate Copied and Shifted 3 Times

Update: Combine to New Name

“Combine to New Name” appends all the geo files associated with the geometries loaded into the workspace into a single geo file. Note that all geometry files being combined must have been updated via overwrite if modified, otherwise the original geometry will be used for combination instead of the modified geometry. After the geo files are concatenated into a single file, the `geoconvert.m` function is run on the new geo file to produce new Delaunay geometry files. Figure 3.27 shows the plates from Figure 3.26 combined and copied twice to form multiple columns of plates.

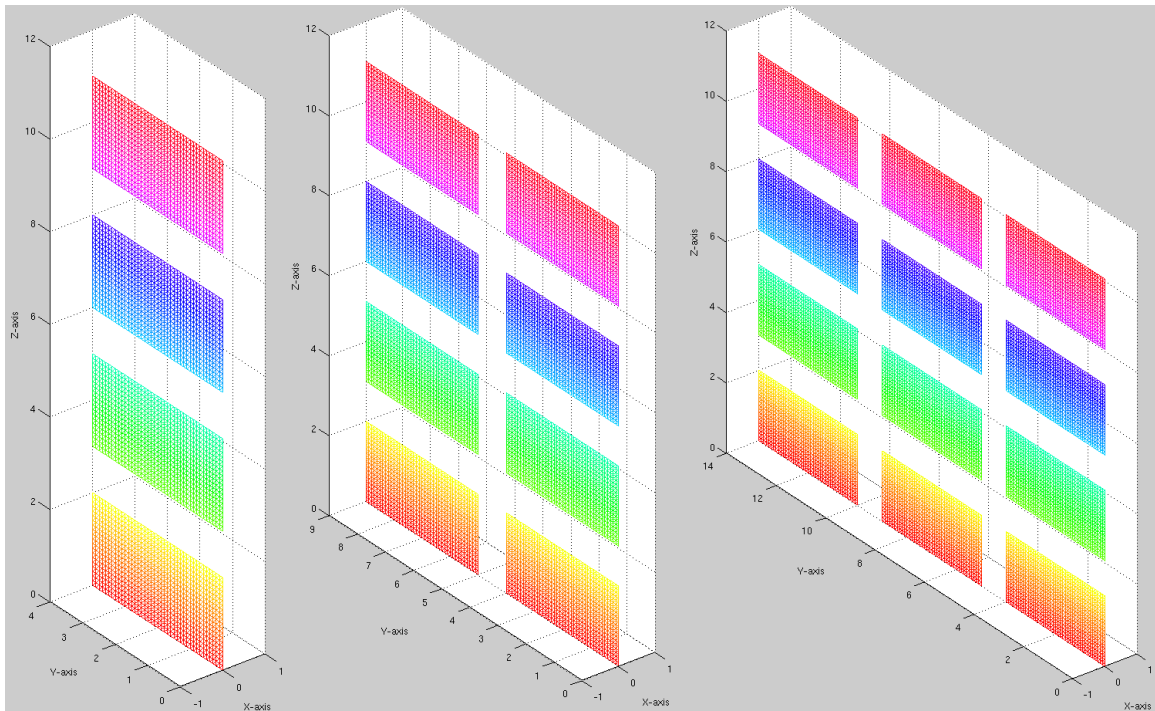


Figure 3.27: Plates Combined to Form Columns

3.3.6 Element Remover

In general, the remover identifies all vertices that lie within a user-selected domain in 3D space and removes any elements that utilize those vertices. Making the job simpler is the fact that the vertices within the vertex.dat file are identified by the row number they occupy. The vertex array is then searched for values within the removal array and output the row numbers they appear on. The element files are then scanned and any elements using the removed vertices are deleted from the elements array and then re-saved to the appropriate element file. Note that the vertex.dat file is never saved over, meaning that the removed vertices still exist. This features allows the vertices to be re-used for creation of new elements using the element joiner discussed in Section 3.3.7. When the geometry is updated in preparation for modeling, the georevert.m function creates elements directly from the element files, using the vertex file for reference only. Any vertices not referenced are not included in the output interface geometry file and are effectively deleted. Figure 3.28 shows the layout of the element remover portion of the main GUI.

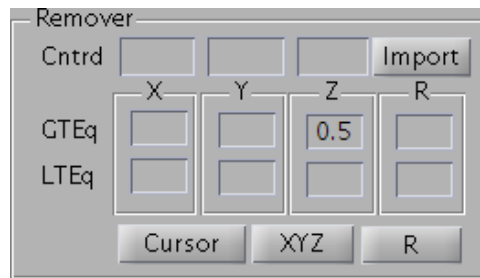


Figure 3.28: Element Remover

Cartesian Removal

The Cartesian Remover provides six planar values that can be used to create a removal domain. The vertex.dat file is first opened and, depending on the settings, vertices found to lie either inside or outside of the specified domain are identified for removal. All element data files associated with the selected geometry are opened and searched for elements using one of the vertices marked for removal. Individual element rows found to contain such vertices are deleted and the element data files are re-saved. The left side of Figure 3.29 shows the results of removing all vertices found to be within the domain $x \leq 0$, $y \leq 0$, and then found to be in the domain $z \geq 0$ in a second call to the remover. The right side of Figure 3.29 shows that element removal of vertices in the domain $x \leq 0.25$, $y \leq 0.25$, $z \geq 0.25$ results in removal of a triangular patch. Swapping the rows of the remover arguments would result in the compliment of the domain being removed, effectively leaving the triangular patch and removing all others.

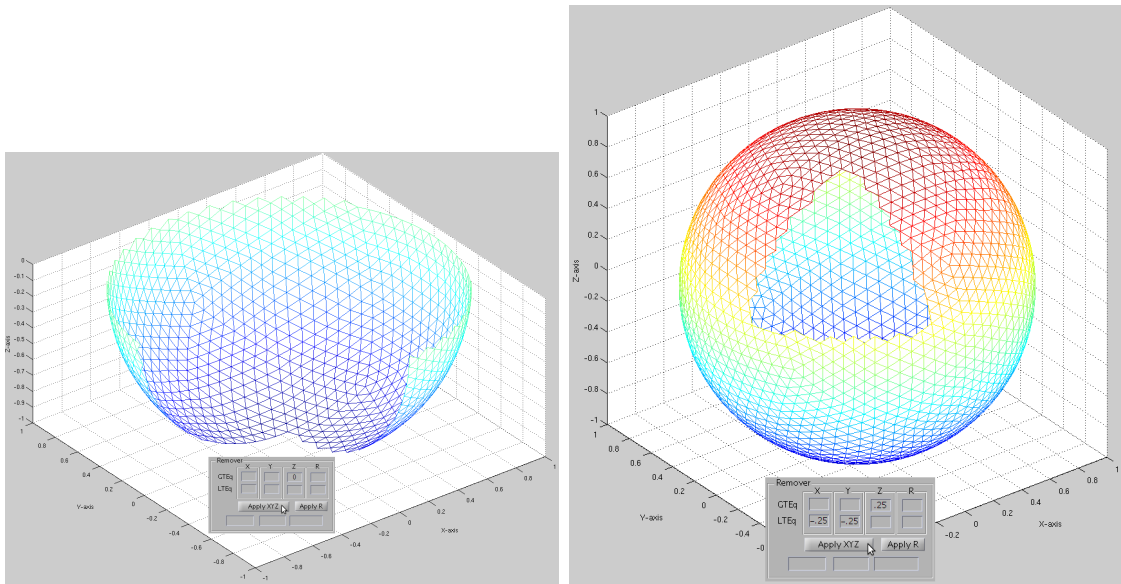


Figure 3.29: Cartesian Removal of Sphere Elements

Spherical Removal

The spherical removal function allows the user to remove elements containing vertices within a specified distance from the given centroid. The centroid coordinates can either be entered manually or the user can use the data cursor capability within the geometry viewer to select a single point and then click the Import button to have the coordinates automatically entered. The vertex.dat file is then opened and all x,y,z values shifted per the centroid coordinates. The magnitude of the adjusted x,y,z coordinates is taken and the radial limits applied. If a value is placed in the greater-than-or-equal-to (GTEq) row, then all vertices identified as falling outside the specified radius are marked for removal. If the radius value is placed in the less-than-or-equal-to (LTEq) row, then all vertices identified as falling within the specified radius are marked for removal. Each element data file pertaining to the geometry selected in the workspace is sequentially opened and searched for elements containing the vertices marked for removal. The rows of elements meeting the search criteria are returned and eliminated. At this point each element data file is re-saved since they contain only elements comprised of vertices not meeting removal criteria.

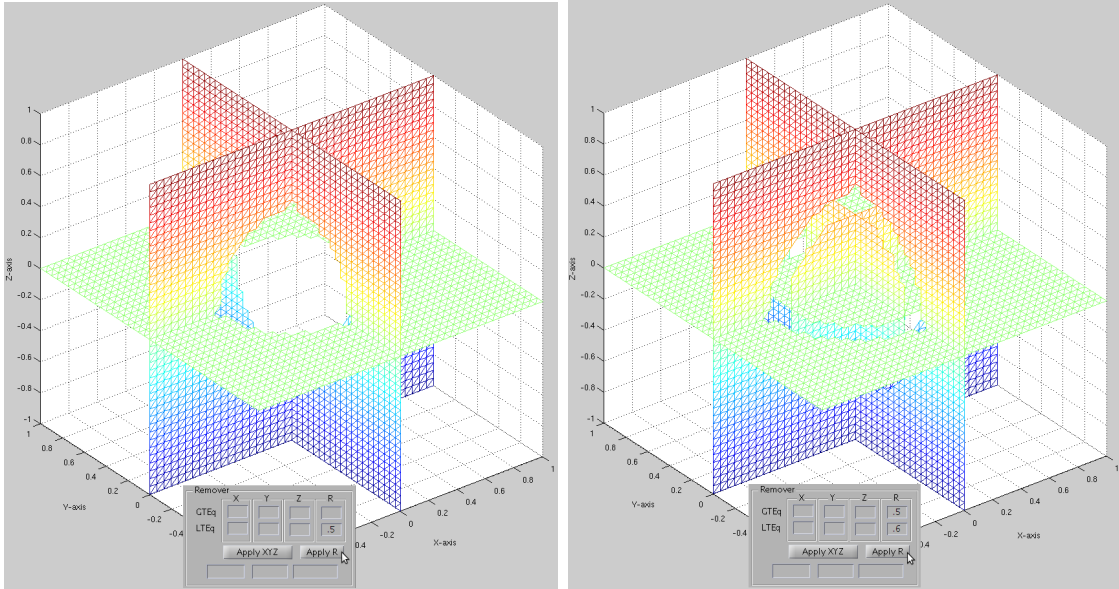


Figure 3.30: Spherical Removal of Retro-reflector Elements

Cursor Point Removal

When working with the geometry viewer, the user has the option of identifying vertices for removal using the data cursor feature of the figure. When the desired vertex is located and selected using the data cursor, the user can press the Cursor button to remove all elements connected to that vertex. Clicking on the cursor button opens the vertex.dat file and shifts all row values by the coordinates of the vertices selected in the geometry viewer. The magnitude of the adjusted x,y,z coordinates is taken and the minimum value located. This value represents the minimal radial distance between the selected point and the available vertices and effectively locates the row of the vertex closest to the selected coordinates. Each element data file is then opened and a list made of all elements found to contain the vertex marked for removal. The row containing reference to the vertex marked for removal are themselves removed and the elemental data files are then re-saved.

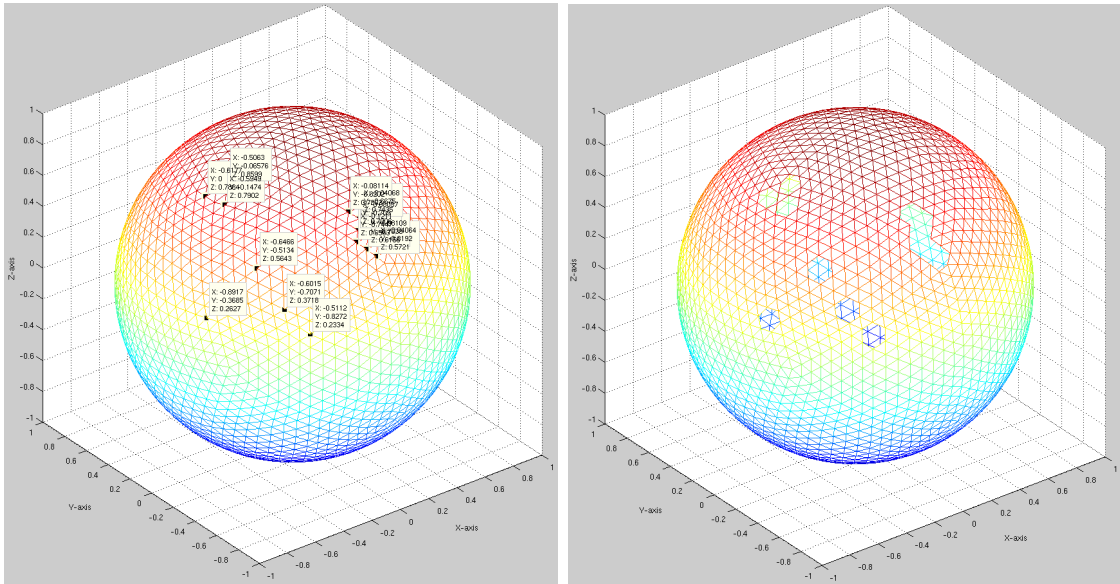


Figure 3.31: Spherical Removal of Retro-reflector Elements

3.3.7 Element Joiner

To create complex geometries from basic geometries it can be necessary to manually join two geometries together. The Joiner block portion of the main GUI, shown in Figure 3.34, allows the user to create new vertices at designated x-, y-, and z-coordinates and to then use these new vertices to create new elements. The Joiner block also allows the user to select already-existing vertices directly from the data cursor feature of the geometry visualization and create new elements using them.

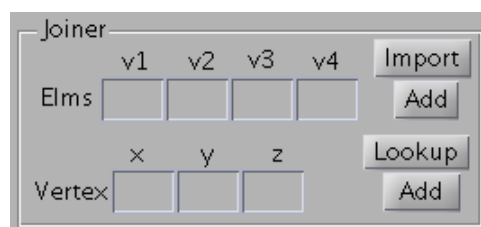


Figure 3.32: Element Joiner

Add Vertex

As geometries are joined together, the necessity of creating new, intermediate vertices sometimes arises. If multiple vertices need to be added, the user has the option of adding the x,y,z coordinates directly to vertex.dat and recording the rows of the newly created vertices. This method can prove especially useful in conjunction with MATLAB generation of points. Vertices created in this manner can then be selected using the data cursor feature from within the geometry viewer.

Add Segments, Triangles and Quads

Elements can also be added to the appropriate .dat file depending on the number of vertices selected by the user. This function is also facilitated by the user's ability to identify points within the geometry viewer using the data_points tool to identify

two², three, or four vertices from which to form a new element. The vertices selected are compared to the vertex.dat file to check for existence and new entries are created if found to be unique. The row containing each vertex of interest is then added to the appropriate elms_*.dat file. If multiple points need to be entered as xyz coordinate sets, the points can be added directly to the geo file and geoconvert called from the command line. Calling the geoconvert function will create vertices and element data files based on all element sets found in the geo file (including the newly added elements). Figure 3.33 shows three vertices being selected with the data_cursor and subsequently joined into a triangular element.

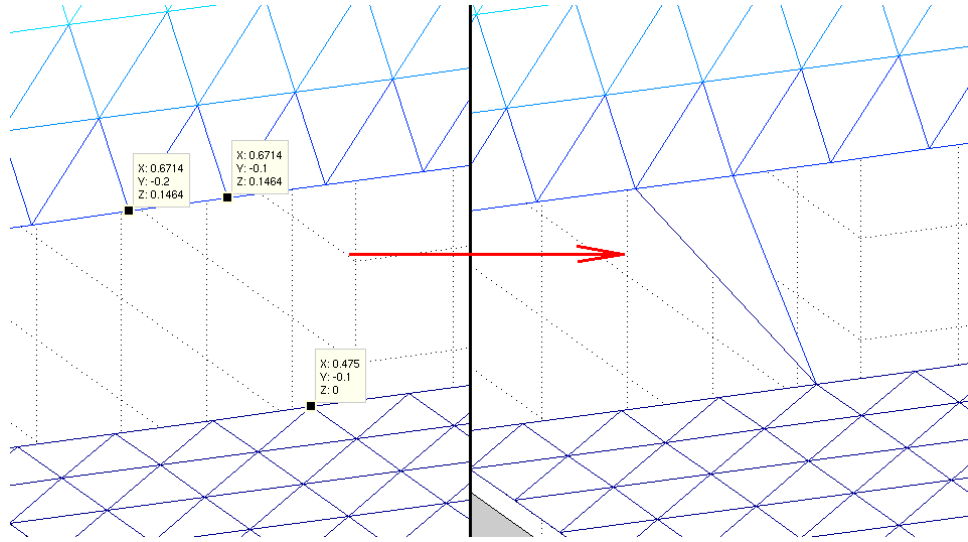


Figure 3.33: Triangular Element Creation

²Segments are created with radius = 0.001λ and resistance = 0Ω .

3.3.8 System Modeler

The system modeler GUIs pass interface geometry files and user settings to the modeler executables and then process and relocate the resultant model files. The “radiate” and “scatter” modeler executables are called from GUIs available through the “model” dropdown of the main GUI toolbar. Both modeler GUIs contain the common settings shown in Figure 3.34. Common attributes are broken down into five areas: Geometry, Solver, Preconditioner, System, and Incident Angle Range. The user first selects a geometry from the dropdown menu at the top of the GUI. This dropdown menu contains all geometries listed under the current project folder. Attributes for the Solver, Preconditioner, and System can then be changed from default values; however, the function of these parameters are outside this scope of this thesis. Additional information can be found in Barrett [5].

The screenshot displays a software interface for configuring system modeler settings. It is divided into three main sections: SOLVER, PRECONDITIONER, and EM_METHOD. The SOLVER section includes a dropdown for SOLVER_TYPE (set to GMRES), and input fields for Iterative Tol (.001), Max Iter (280), Restart (280), and a Trace checkbox (set to false). The PRECONDITIONER section features a dropdown for PRECON_TYPE (set to None), and input fields for Drop Tol (.001), Fill Factor (1), Int Dist (.3), Pivot Factor (1), and a dropdown for Precon Side (set to RIGHT). The EM_METHOD section contains radio buttons for EM_TYPE (MLFMA selected, MM unselected), dropdowns for Levels (1) and Sig. Digits (3), a dropdown for Matching (NORMAL), radio buttons for Optimize for (SPEED selected, MEMORY unselected), a Correct IBC checkbox (set to false), input fields for Quad. Order (6) and Group Size (0.25), and a text field for Model Name (test_model). At the bottom right are Model and Reset buttons.

Section	Parameter	Value
SOLVER	SOLVER_TYPE	GMRES
	Iterative Tol	.001
	Max Iter	280
	Restart	280
	Trace	false
PRECONDITIONER	PRECON_TYPE	None
	Drop Tol	.001
	Fill Factor	1
	Int Dist	.3
	Pivot Factor	1
	Precon Side	RIGHT
EM_METHOD	EM_TYPE	MLFMA
	Levels	1
	Sig. Digits	3
	Matching	NORMAL
	Optimize for	SPEED
	Correct IBC	false
	Quad. Order	6
	Group Size	0.25
Model Name		test_model
Buttons		Model, Reset

Figure 3.34: Common Portion of System Modeler GUIs

Scatter Model GUI

Monostatic scatter modeling, or backscatter modeling, illuminates the geometry with a plane wave source from multiple user-defined incidence angles and measures the resultant re-radiated field intensity. The fields produced by the plane wave source induce surface currents on the geometry which, in turn, re-radiate fields of their own that are measured as backscatter. The source is relocated for each incident angle and the system is solved for each scattering angle. The field intensity measured at each scattering angle produces a backscatter pattern showing how strongly the geometry re-radiates when illuminated from that angle. The scatter model GUI accepts input of illumination angles by definition of start, stop, and step values in both θ and ϕ . After specifying the desired model name, pressing the “Model” button calls the backscatter model executable, creates the new model folder, and then processes and relocates the resultant model files from the MLFMA directory to the newly created model directory. Figure 3.35 shows the top portion of the backscatter GUI. Due to the time investment required to solve the system multiple times, the user is encouraged to judiciously select the observation angles and step size employed. Selecting too few observation steps can lead to poor backscatter pattern resolution and potential omission of sidelobes. Bistatic scattering can also be accomplished using the scatter model GUI.

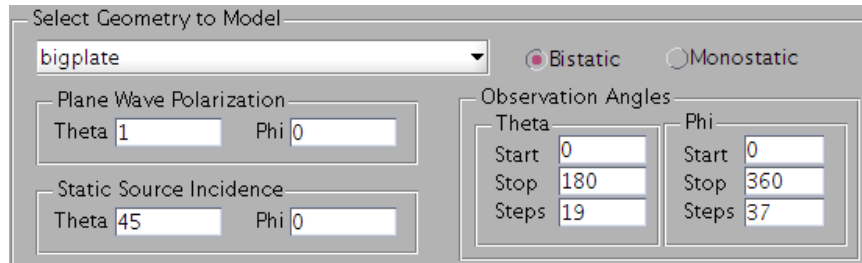


Figure 3.35: Backscatter GUI

Radiation Model GUI

Radiation modeling allows a field source to be introduced as a plane wave, a voltage on a geometry patch or segment, or an infinitesimal (Hertzian) dipole in space. Each type of source is introduced by clicking the appropriate button: add plane wave source, add delta source, or add Hertzian dipole. Each button opens a small GUI (see Figure 3.8) that accepts user input describing each type of source. A plane wave source requires the user to identify the angle of incidence in θ and ϕ and the plane wave polarization as a vector combination of θ and ϕ components. A delta source requires a position in Cartesian

coordinates and the real and imaginary components of the complex voltage be given. The segment closest to the Cartesian coordinate point is selected as the source segment and the complex voltage applied to it. A Hertzian dipole requires a position given in Cartesian coordi-

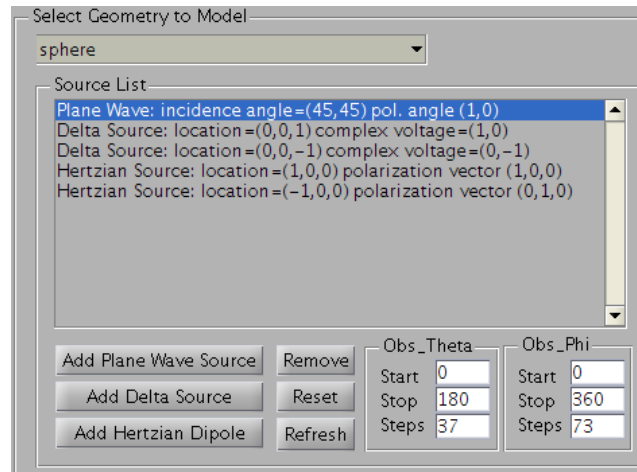


Figure 3.36: Source Model GUIs

nates as well as a dipole orientation given as a vector combination of unit vectors in the x-, y-, and z-direction. Individual sources can be removed from the “src” file by selecting them in the source list and then pressing the “remove” button. Pressing the “refresh” button will reload the source list from the “src” file and pressing the “reset” button will delete the “src” file. The radiation intensity is then measured at multiple user-defined observation angles and the results plotted to produce a radiation pattern. In contrast to backscatter modeling, radiation modeling only needs to be solved once since the source does not change between observation measurements. The default step size of the radiation model is five degrees in both θ and ϕ .

3.3.9 Data Visualization

The data visualization function block is responsible for converting raw data into graphical representations and can only be invoked by interaction with the GUI workspace. Geometry visualization occurs when a new geometry is loaded to the workspace, a geometry is selected within the workspace, or as a companion figure whenever a surface current visualization is requested. Geometry visualization occurs in Figure 1 and is discussed in Section 3.3.9. Surface current visualization occurs when an entry in the current viewer dropdown menu is selected or the Up/Down buttons are pressed. Surface current visualization occurs in Figure 2 and is discussed in Section 3.3.9. Field pattern visualization occurs when one of the three field visualization buttons is pressed. Field visualization occurs in Figure 3 and is discussed in Section 3.3.9. Table 3.5 lists the figures and files associated with each viewer.

Figure	Title	Files Visualized
1	Geometry Viewer	vertex.dat, elms_line.dat, elms_tri.dat, elms_quad.dat
2	Surface Current Viewer	curr#, SC#.dat, currheader
3	Field Pattern Viewer	fiel

Table 3.5: Viewer Types

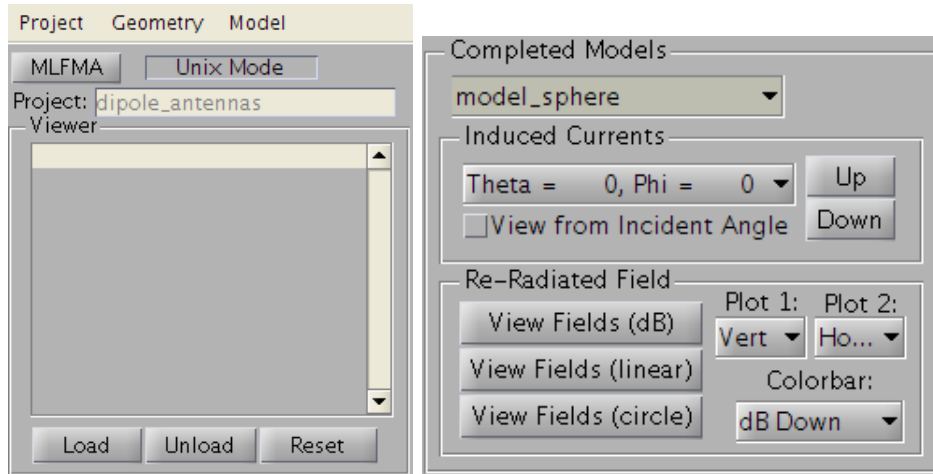


Figure 3.37: GUI Workspace, Completed Models, and Surface Currents

Geometry Viewer

The geometry viewer is responsible for generating mesh geometric representations from Delaunay geometry files whenever a selection is made in the GUI workspace (Figure 3.37). The geometry viewer systematically explores every geometry folder present in the workspace and plots all el-

ement files found. The resultant shape data is displayed in the geometry viewer shown in Figure 3.38. Element data for every geometry in the workspace is plotted together, giving the impression that all geometries in the workspace are available for modification or modeling as a single geometry. Users are encouraged to “combine” geometries when this result is desired, since only a single geometry file can be modified or modeled at a time.

MATLAB figure tools allow the user to

interact with the shape by zooming in, zooming out, moving, rotating, or selecting data points. The Geometry Viewer also serves as the primary avenue by which the user interacts with the element joiner and element remover function blocks. To bypass the geometry viewer in these instances would require the user to manually modify the Delaunay geometry files instead. It must be stressed that colors in the geometry viewer represent height in the z-direction only.

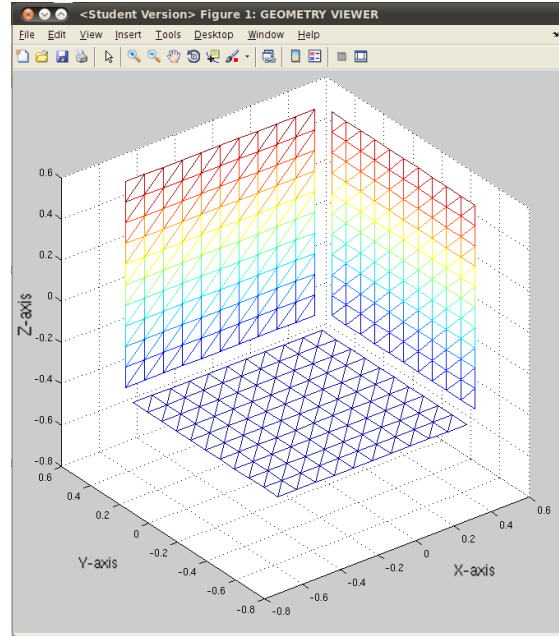
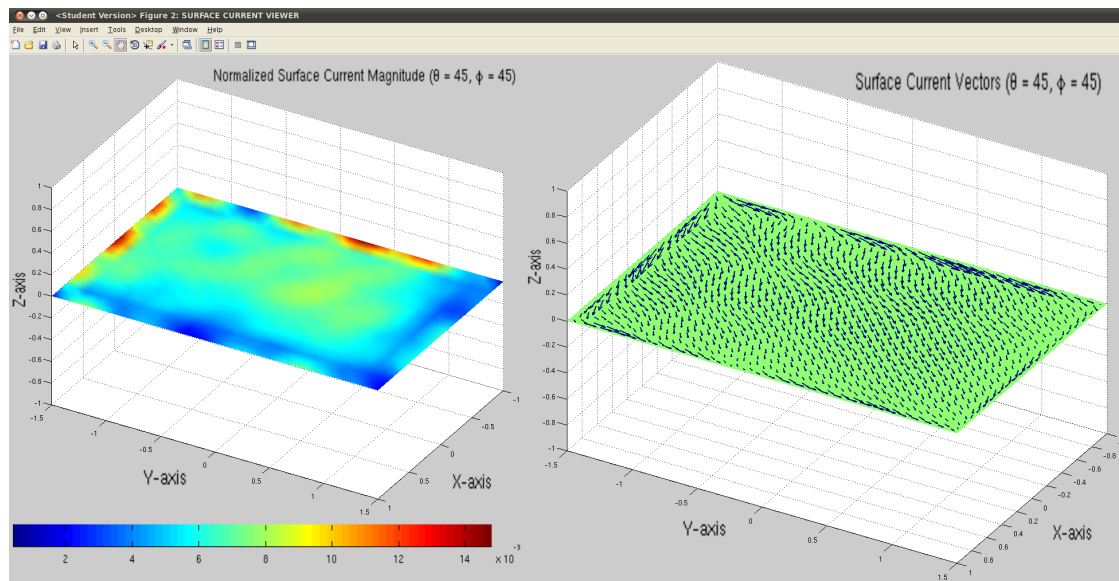


Figure 3.38: Geometry Viewer

Surface Current Viewer

Whenever a selection is made within the current viewer (Figure 3.37), the surface current files corresponding to the observation angle are loaded, the surface current magnitude at each vertex is plotted on the left, and the surface current vector on each element is plotted on the right. The surface current magnitudes are mapped to representative colors to visually demonstrate the current distribution over the surface of the geometry. The result is very similar to the geometry viewer output, with two differences: the mesh geometry has been replaced with patch elements and the colors represent the surface current magnitude instead of the elevation in z . Filling the checkbox marked "View from Incident Angle" changes the plot view to that of the incident angle. The surface current viewer is shown below in Figure 3.39.



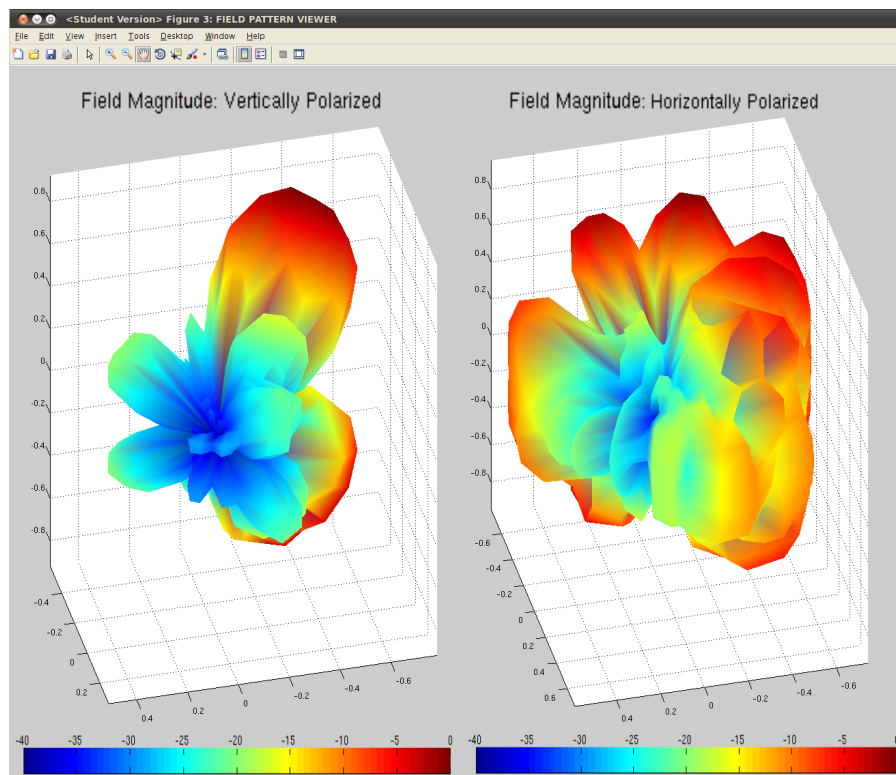
(a) Magnitude

(b) Direction

Figure 3.39: Surface Current Viewer

Field Pattern Viewer

When a model is located within the selected geometry folder, pressing any of the three visualization buttons produces a plot of the field magnitudes. This data is stored in the “fiel” file and its representation differs depending on which button is pressed. Pressing the “dB” button displays the relative field strength scaled to between 0 and -50dB. Pressing the “linear” button displays the absolute field strength, resulting in large returns in directions orthogonal to flat surfaces. Pressing the “unit circle” button projects the field magnitude onto a sphere of unit radius. The user can also elect to display the actual field magnitude values by making the appropriate selection from the dropdown menu. In Figure 3.40, the left plot displays the pattern of the vertically polarized field component and the right plot displays the pattern of the horizontally polarized field component.



(a) Vertically Polarized

(b) Horizontally Polarized

Figure 3.40: Field Pattern Viewer

CHAPTER 4

Geometry Modeling

Before modeling any complex geometries, it is important to establish a good relationship between GUI output and theoretical output of common antenna structures. To achieve this, several basic geometries have been modeled and compared to their well-known radiation patterns. Several structures have been selected for modeling: a sphere, a yagi antenna, a planar plate, several dipoles, a helical antenna, and a 3D cube. In the following sections, the procedure for building and modeling each geometry is discussed. Afterward, results are presented and compared to theoretical expectations. When creating any significantly complex piece of software it helps to have examples to follow. The examples below fully show the capabilities of the MLFMA GUI and its associated software components.

4.1 Backscatter Models

Backscatter Modeling provides an indication of which incident angles produce the strongest returns for a given geometry. This is of primary importance when the geometry serves to produce strong signals in all directions (as is the case of a retro-reflector). On the other hand, if a geometry is meant to have low radar visibility, backscatter modeling provides insight into illumination directions that provide unacceptably large radar cross sections.

4.1.1 Sphere

The first geometry to be built and modeled is a basic sphere. To begin, the MLFMA GUI is loaded in MATLAB by first changing the working directory to the MLFMA GUI directory and then executing the “mlfma” command from the command prompt. In Figure 4.1 the path is displayed at the top of the window and the appropriate command is being called from the command line at the bottom. The Project Loader

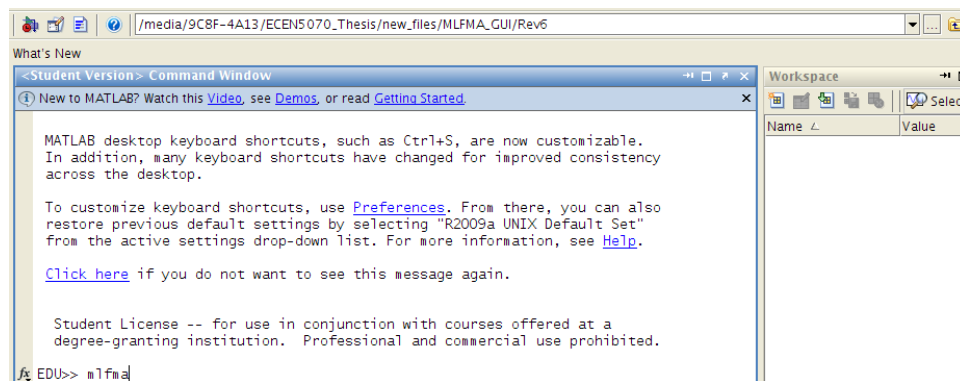


Figure 4.1: Loading the GUI from the MATLAB Command Line

GUI appears pre-loaded with all currently available projects for user-selection. As this is the first basic geometry to be modeled, a new project folder will need to be created.

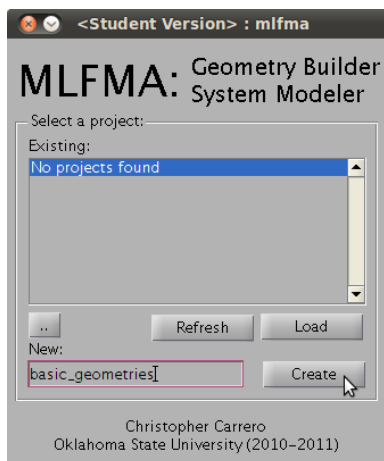


Figure 4.2: Create Project

The Project Loader GUI is automatically closed and the Main GUI is opened in the top-left corner of the screen with the name of the new project folder identified in the top-left corner of the GUI.

No geometries yet exist in the `basic_geometries` project folder, so a new geometry will need to be built. To build the sphere geometry, the user clicks on the Geometry entry in the toolbar at the top of the main GUI and then the sub-entry titled

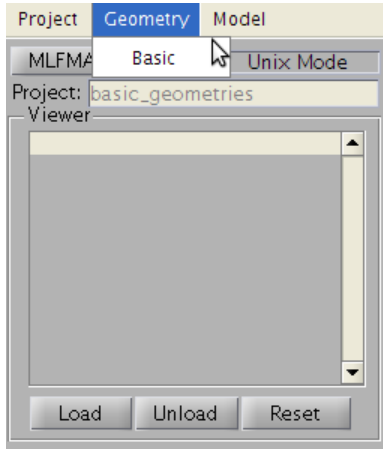


Figure 4.3: Toolbar Element

“Basic”, as shown in Figure 4.3. This loads the Basic Geometry Builder GUI and commands the main GUI to wait for the Build GUI to finish its task. From the Basic Builder GUI, the user can select the desired geometry from a list of choices in the dropdown menu at the top of the GUI. To build a sphere, the `sphere_tri(R,C)` option is selected in the dropdown menu. The `sphere_tri` geometry requires the designation of a centroid vertex and radius and then creates

a sphere comprised of triangular elements, all of which have dimensions not exceeding `ideal_del`. Once the user-attributes are selected, a unique name must be given to the geometry before the “Create” button is pressed. After the builder subroutine finishes creating the geometry, the `geoconvert.m` function converts the “geo” file to `vertex.dat` and any necessary `elm*.dat` files. Figure 4.4 shows the progress bar that is displayed to notify the user of the relative time remaining before file conversion is complete.

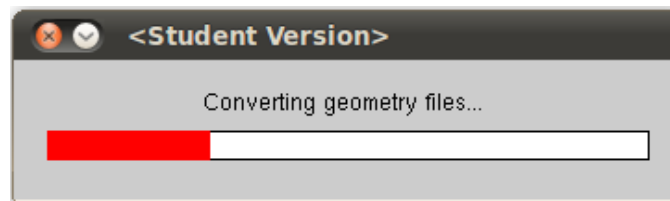


Figure 4.4: `geoconvert.m` Progress Bar

When the files conversion process is complete the geometry will automatically appear in the workspace and the geometry viewer will appear, displaying a mesh representation of the new geometry along with any other geometries present in the workspace. If other geometries are present, the user can remove them by highlighting their name in the workspace and pressing the “Remove” button. Alternately, the

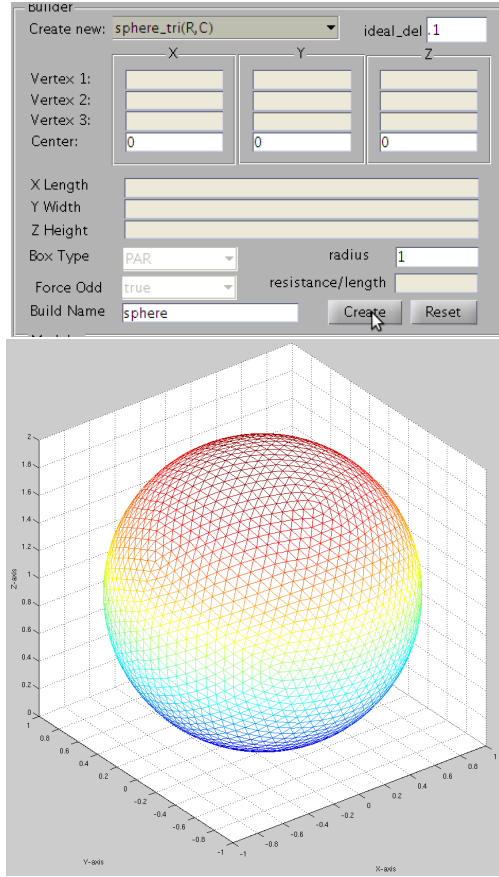


Figure 4.5: Sphere Geometry

“Reset” button can be pressed to unload all geometries present in the workspace. Figure 4.5 shows the sphere build parameters and the resultant sphere created by the build process. With the sphere geometry created, the user has the option of creating a model based on the geometry. The sphere geometry is selected from the dropdown and then the available model attributes can be adjusted to give the user the desired environment. Once the user-parameters are set, a unique model name will need to be entered into the model name edit box at the bottom of the GUI before the “Model” button is pressed. Default values are selected for modeling and incident

angles of θ and ϕ are selected to range from 0° to 180° in 10° increments and from 0° to 360° in 20° increments, respectively. Although model creation can take a significant amount of time, the user will be able to see the mlfma code outputting to the MATLAB command window. This information is the basis for the “modellog” log file.

Once the model subroutine is complete, the resultant surface current and field pattern files are copied from the MLFMA directory to the newly created model folder along with the “geo”, elms*.dat, and vertex.dat files. From there, the curr files containing the complex currents at each vertex are converted to SC#.dat files after redundancies are removed. At this time, the “Completed Models” portion of the workspace is updated to reflect any model folders found within the geometry folder. Likewise, the “Induced Currents” portion of the workspace is updated with all current

files found within the selected model folder. In Figure 4.6, the “model_sphere” model folder is selected and the induced surface current files resultant from a field incident at $\theta = 0$ and $\phi = 0$ are loaded. The induced current files are all identified by the plane-wave incidence-angle responsible for their creation. Clicking

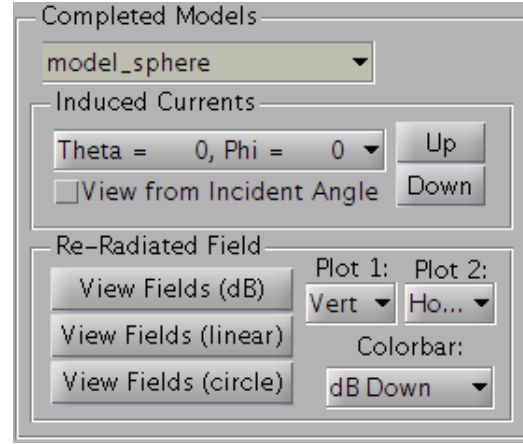
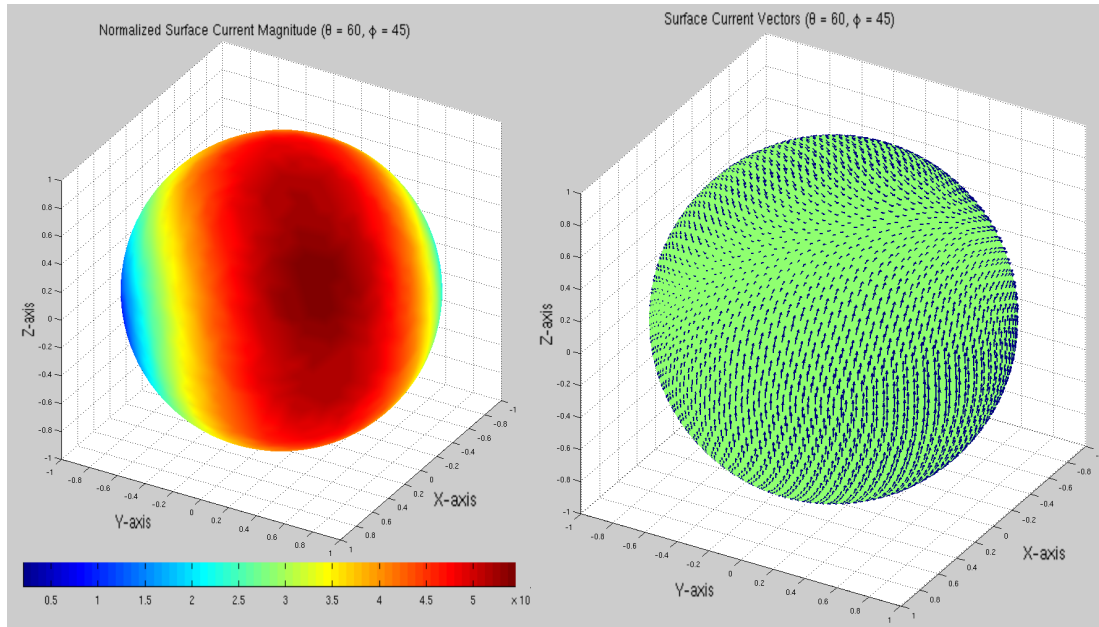


Figure 4.6: Models and Currents

on the listbox and selecting an incidence angle will cause the surface current viewer to appear with the corresponding current files loaded into the viewer. Figure 4.7 shows the result of clicking on the entry displayed in the “Induced Currents” listbox. In this instance, the incidence angle is 60° in θ and 45° in ϕ . The left subplot displays



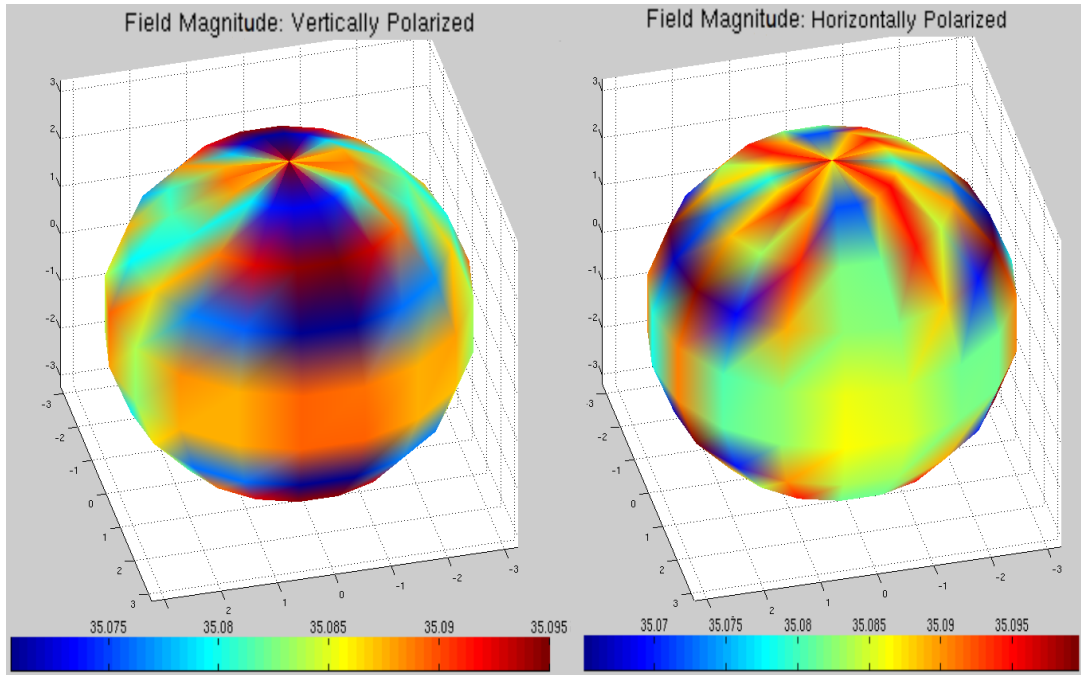
(a) Magnitude

(b) Direction

Figure 4.7: Sphere Surface Currents

surface current magnitudes at each vertex and the right subplot displays surface current vectors present on each element. Clicking the “Up” and “Down” buttons next to the currents listbox allow the user to move between adjacent current files. Each

time the button is clicked, the surface current viewer redraws the surface currents according to the current .dat files associated with the listbox entry. The currheader file is responsible for maintaining the relationship between the surface current filenames and the incidence angle responsible for their creation. By selecting the “View from Incident Angle” checkbox, all induced surface current plots are auto-viewed from the direction of the incoming plane wave instead of the default viewing angle. Finally, the user has the option of viewing the actual field magnitudes resultant from all the incidence angles processed during modeling. In the instance of a sphere, the field pattern is expected to be entirely independent of the incidence angle. As expected, the field magnitude values of Figure 4.8 show minimal directional-dependence and extreme values differ by only 0.03.



(a) Vertically Polarized

(b) Horizontally Polarized

Figure 4.8: Sphere Backscatter Pattern

4.1.2 Plate

The second geometry to be built and modeled is a planar plate. Development of a plate geometry follows the same procedure as that of Section 4.1.1. If the GUI has just been loaded from the MATLAB command line the basic_geometries project folder is loaded from the Project Loader GUI. Otherwise it can be loaded directly from the Main GUI. Build Basic Geometry is selected from the toolbar and the `plate_tri(L,W,C)` geometry is selected from the dropdown. The `plate_tri` geometry requires the user to define the centroid, length in the x-direction, and width in the y-direction. Figure 4.9 shows the geometry will be named “plate” and will have a length of $x = 2\lambda$, $y = 3\lambda$, and will be centered on the origin. Figure 4.9 displays the geometry build parameters and the resultant geometry built at $z = 0$ in the x-y plane. Modeling is accomplished by selecting the “Scattering” option from under the “Model” toolbar element. The GMRES solver is again used and, to achieve good resolution, θ and ϕ are selected to range from 0° to 180° and 0° to 360° , respectively, in 10° increments. The resulting surface current plots show, as expected, that the currents are strongest when the source is orthogonal to the plate and are weakest when the source is parallel to the plate. Figure 4.10 shows the induced surface currents for $\theta = 45^\circ$, $\phi = 45^\circ$. The resultant field pattern plots indicate that the strongest response occurs directly above or below the plate. Additionally, for vertically polarized incident plane waves the re-

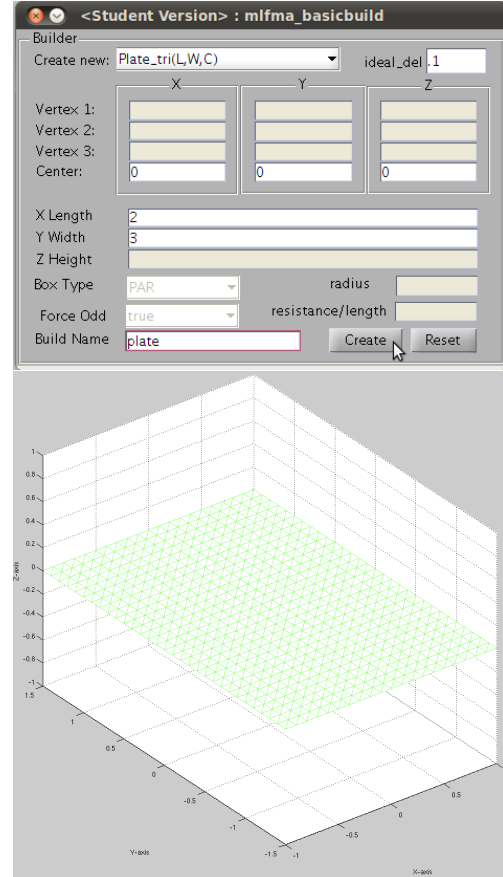
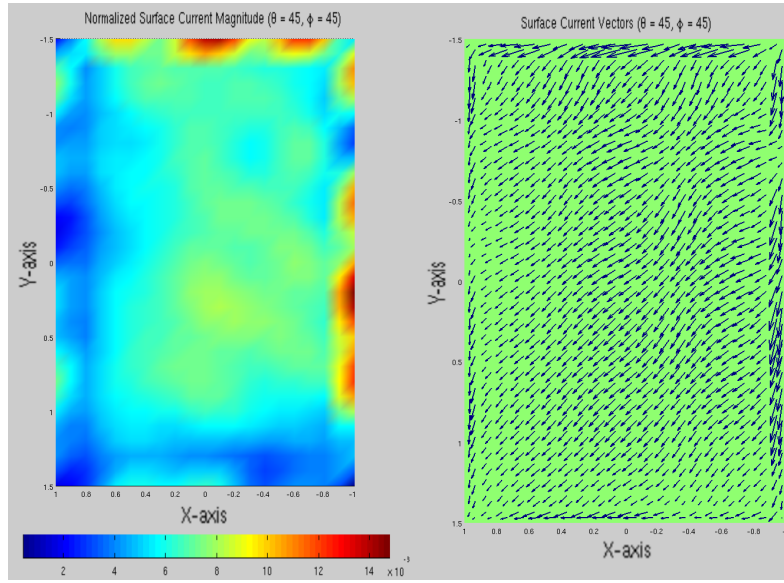


Figure 4.9: Plate Build

The GMRES solver is again used and, to achieve good resolution, θ and ϕ are selected to range from 0° to 180° and 0° to 360° , respectively, in 10° increments. The resulting surface current plots show, as expected, that the currents are strongest when the source is orthogonal to the plate and are weakest when the source is parallel to the plate. Figure 4.10 shows the induced surface currents for $\theta = 45^\circ$, $\phi = 45^\circ$. The resultant field pattern plots indicate that the strongest response occurs directly above or below the plate. Additionally, for vertically polarized incident plane waves the re-

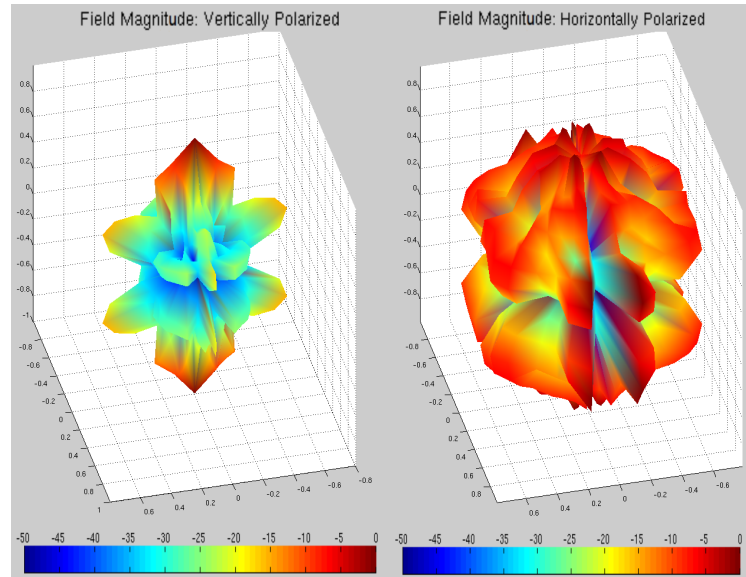


(a) Magnitude

(b) Direction

Figure 4.10: Plate Surface Currents

sponse is almost null at incident angles of around $\theta = 90^\circ$. Horizontally-polarized incident plane waves do not have the same near-null response and appear unaffected by the minimal available surface area. Figure 4.11 shows the field patterns for both vertically and horizontally polarized backscatter.



(a) Vertically Polarized

(b) Horizontally Polarized

Figure 4.11: Plate Backscatter Pattern

4.1.3 Cube

The third geometry to be built and modeled is a basic cube with faces parallel to the xy-, xz-, or yz-plane. The cube geometry is created by selecting the `cube_tri(L,W,H,C)` entry from the build geometry drop-down menu. The `cube_tri` geometry requires the user to define the length, width, height, and centroid of the desired cube to be created. Figure 4.12 shows the user parameters selected to build the plate geometry.

The far field model used for this geometry again uses the GMRES solver and values of θ and ϕ ranging from 0° to 180° and 0° to 360° in 10° increments. The resultant surface current plots show that the strongest surface currents occur when the source is orthogonal to one of the planar sides. Also, when viewing the back of the cube and stepping through the different incident angles it is possible to see the formation of creeping waves on the back of the geometry. Figure 4.13 shows the surface currents induced by a vertically polarized plane wave incident at $\theta = 45^\circ$ and $\phi = 90^\circ$. The resultant field intensities are strongest when the source is orthogonal to one of the sides. Figure 4.14 shows the field patterns for vertically and horizontally polarized backscatter due to that same incident plane wave. The red portions of the vertically polarized backscatter indicate that the response in the direction of the flat surfaces is 10dB stronger than in any other direction. The field pattern also shows that diagonal backscatter angles have a sinusoidal response as constructive and destructive interference occurs.

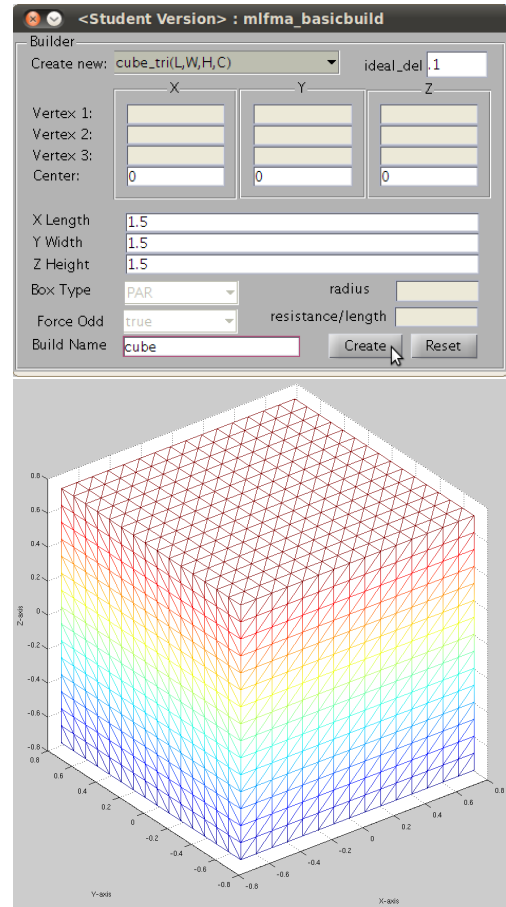
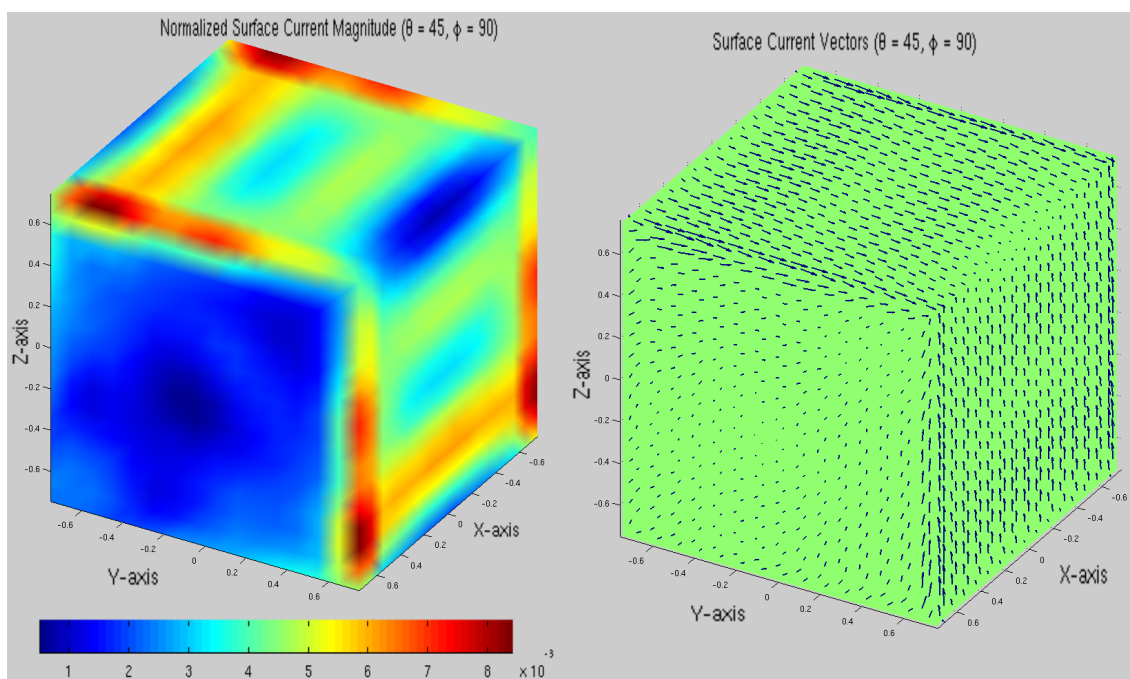


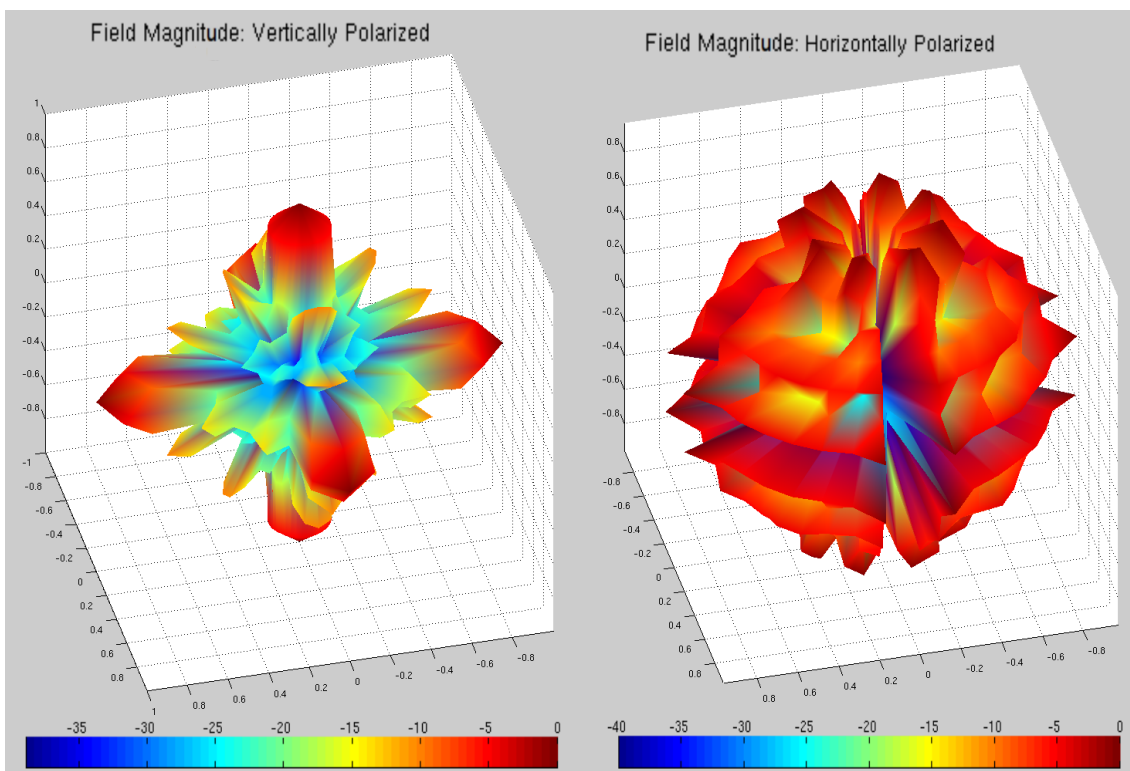
Figure 4.12: Cube Build



(a) Magnitude

(b) Direction

Figure 4.13: Plate Surface Currents



(a) Vertically Polarized

(b) Horizontally Polarized

Figure 4.14: Cube Backscatter Pattern

4.1.4 Hemispherical Dipole

The hemispherical dipole consists of two hemispheres (half spheres) connected by a wire. To create the top hemisphere, a sphere of radius = 0.5λ and a square plate of length = λ are created, both with an ideal_del value of 0.1λ . The destructor is called to remove sphere vertices lying in the $z \leq 0$ domain and plate vertices lying in the $r \geq 0$ domain. The resulting geometries are shown below in Figure 4.15. The two geometries are each saved via overwrite and then combined to a new shape

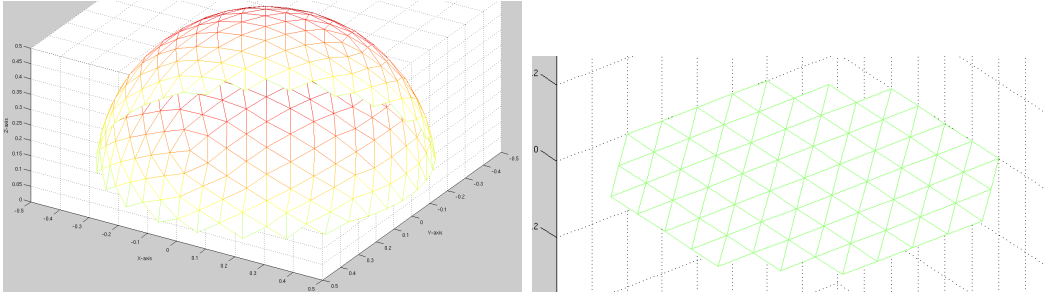


Figure 4.15: Hemisphere Components

named `top_half`. The gap between the two components is filled in using the element joiner portion of the main GUI. The `data_cursor` feature of the geometry viewer is selected and sets of three vertices are identified directly on the part representation. After the vertices of the new triangular element are identified, the import button is pressed to identify the vertices that are selected (indexed rows within `vertex.dat`) and then the add button is pressed to concatenate the new element onto the `elms_tri.dat` file. Figure 4.16 shows the first element that was created in this manner as well as a representation of the geometry after about 90% of the new elements had been created. Once all the new elements have been created the `top_half` geometry is saved and copied to a new geometry named `bottom_half`. To create the gap for the wire, `top_half` is first translated upward by $z = 0.05$ and then `bottom_half` is translated downward by $z = -0.05$. The two geometries are both saved via overwrite and then combined into a new geometry named `hemi_dipole` to produce a common `vertex.dat` file that

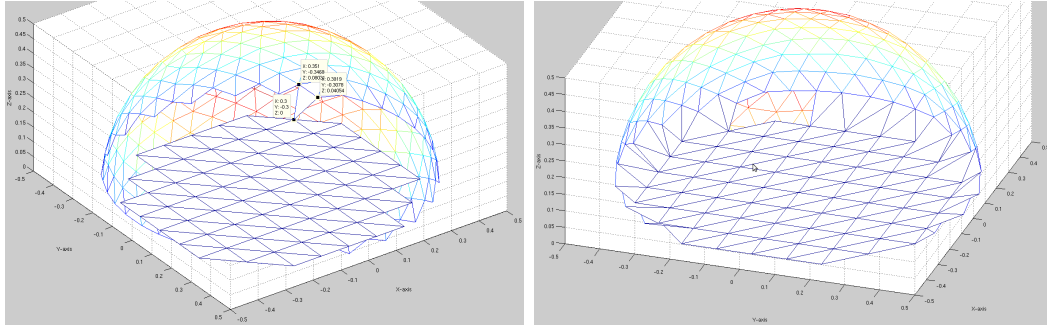


Figure 4.16: Element Creation

the wire element can be added against. The data_cursor feature in the geometry viewer is again employed and the points $(0,0,0.05)$ and $(0,0,-0.05)$ located. These two points form the start and stop vertices for the wire segment to be created. Once the points are located and selected the import button in the element joiner portion of the main GUI is pressed followed by pressing the add button. This creates a new file titled `elms_line.dat` with one entry. Figure 4.17 shows the `top_half` and `bottom_half` geometries before they are separated and the finished geometry. Once again the

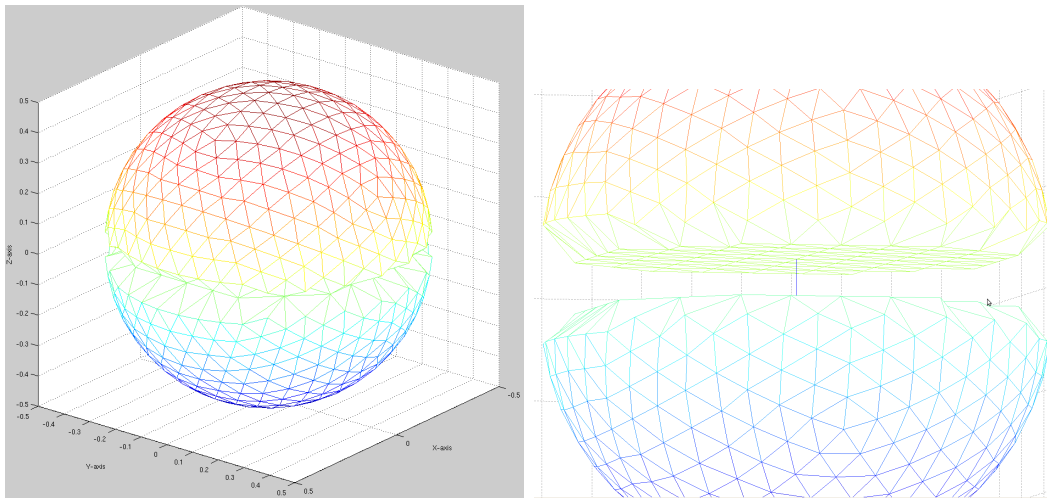


Figure 4.17: Finished Geometry

geometry is saved via overwrite in preparation for modeling. The delta source model GUI is called from the toolbar and a source placed on the segment nearest the point $(0,0,0)$, which can be expected to be the only wire segment in the `elms_line.dat` file.

The modeling proceeds as usual until the `f.s.solve()` function is called from within the C++ model executable. At this time two errors are generated due to the current version of the software attempting to use an incorrect method of attaching a wire to a segment. The error output are provided below in Figure 4.18 and Figure 4.19. This is a known issue and a fix is available in a newer revision of the code.

```
B.E. on the preconditioned system: 0.9222E-03
B.E. on the unpreconditioned system: 0.9222E-03
info(1) = 0
Number of iterations (info(2)): 118
Theta Phi VV VH HV HH
GET_POINTS: Source JOINED_TRIANGLE not yet defined.
ans =
    1
fid =
    6
All elements NOT of same type
RESULT: 748 unique points out of 4466 total.
Elapsed time is 9.156588 seconds.
Elapsed time is 0.203270 seconds.
EDU>>
```

Figure 4.18: Error Output

```
Will cause problems with higher order basis functions and C/MFIE.
WARNING: Edge starts on same vertices for positive and negative element.
Will cause problems with higher order basis functions and C/MFIE.
WARNING: Edge starts on same vertices for positive and negative element.
Will cause problems with higher order basis functions and C/MFIE.
WARNING: ELEMENT: Wire/triangle junctions are inaccurate with point matching
WARNING: ELEMENT: Wire/triangle junctions are inaccurate with point matching
Total nodes = 2234
2234 unknowns.
Setting up level 1 groups.
M[0] = 64
L[0] = 5
K[0] = 50
Level 1 interactions = 2220 out of 4096 possible.
near_interactions = 2515864
2234 unknowns.
2515864 sparse matrix elements out of a total of 4.99076e+06 total elements.
50.4105%
Setting up sparse matrix. (Multiply following by 100,000 to get number found.)
Buffers allocated
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 Sparse fill time = 8.19
reordering CORR_mat
sort M_div_rot
sort M_rot_div
sort M_h
Sparse fill time = 8.59

Done setting up sparse matrix, now converting.
Finalize int_mat
Done converting sparse matrix.
Time = 0.02

Min diag = (1.193,-152.759)
Min diag number = 536
Max diag = (1.49779,-327.366)
Max diag number = 525
Setting up translation matrices
```

Figure 4.19: Error Output Continued

4.2 Radiation Models

Radiation models provide insight into the radiation characteristics of an antenna structure. Of particular importance is the antenna pattern produced by plotting the radiation intensity as a function of azimuth and elevation. In the paragraphs that follow, several well known and oft-analyzed geometries are built and their radiation patterns plotted.

4.2.1 Dipole

The simplest radiating element to model is a basic dipole. Creation of the dipole is carried out in much the same manner as the sphere from Section 4.1.1. After the Project Loader GUI has been called from the MATLAB command prompt, the previously-created “basic_geometries” project is selected from the listbox and then the “Load” button is clicked, as shown in Figure 4.20. The Main GUI window loads

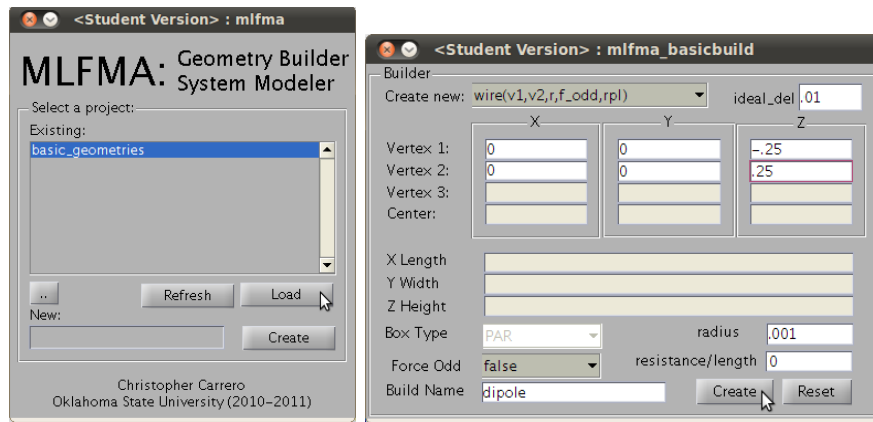


Figure 4.20: Load Project

to the top-left corner of the screen and with the appropriate project displayed in the top-left corner of the GUI. To create the new dipole geometry, the user will click on the Geometry toolbar element and then on the Basic sub-element as before. The appropriate listbox entry for creating the dipole is the `wire(v1,v2,r,f_odd,rpl)` geometry. This geometry requires the user to specify the two vertices to be connected by the wire, the radius of the wire, whether or not to force the number of segments to an odd number, and the resistance-per-length (rpl) of the wire. This will be

a half-wave dipole extending from $(0,0,-.25)$ to $(0,0,.25)$. The wire is created with $\text{radius}=0.001\lambda$, $\text{rpl} = 0\Omega$, and $\text{force_odd} = \text{no}$. The corresponding entries in the Basic Builder GUI are shown in Figure 4.20 and the resulting geometry is plotted in Figure 4.21. The geometry is modeled by selecting “Radiate” from under the Model toolbar element. The GMRES solver is used and θ and ϕ are selected to range from 0° to 180° and 0° to 360° , respectively, each in 5° increments each. The resulting surface current plots exhibit the expected sinusoidal nature of the current on the dipole. Figure 4.22

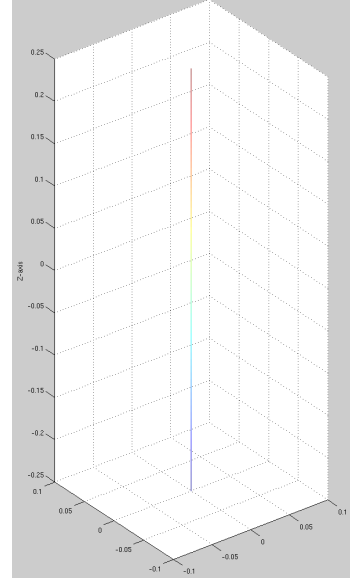


Figure 4.21: Dipole Geometry

shows the surface currents resulting from a delta source at the origin. Only vertically-polarized scattered fields are discernible in field plots. Additional dipoles of length λ , $5\lambda/4$, and $3\lambda/2$ were also modeled. Figure 4.23 compares the resultant radiation patterns. These results show good agreement to those discussed in Kraus [10].

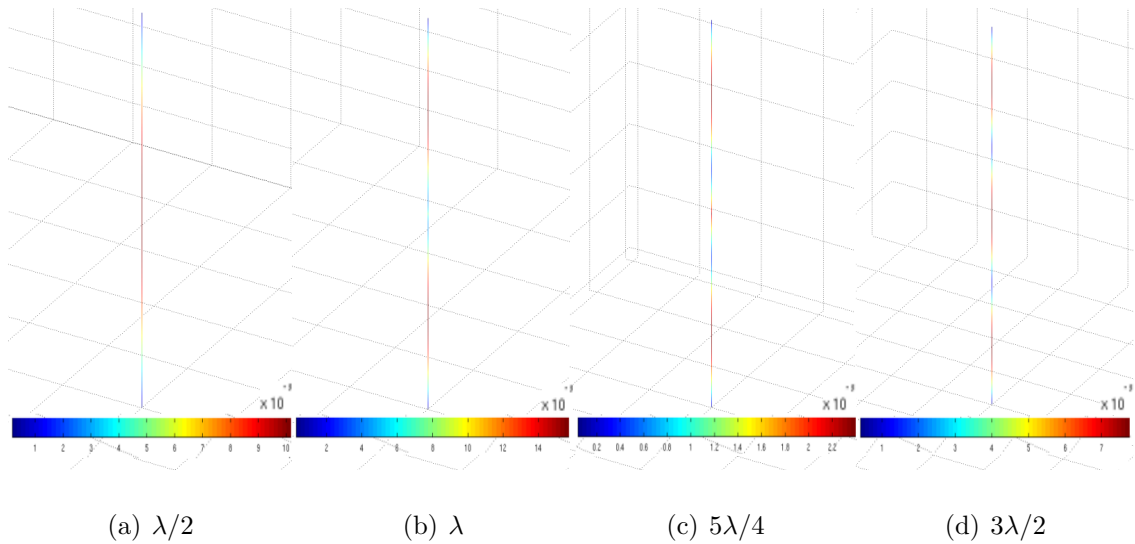


Figure 4.22: Dipole Surface Currents

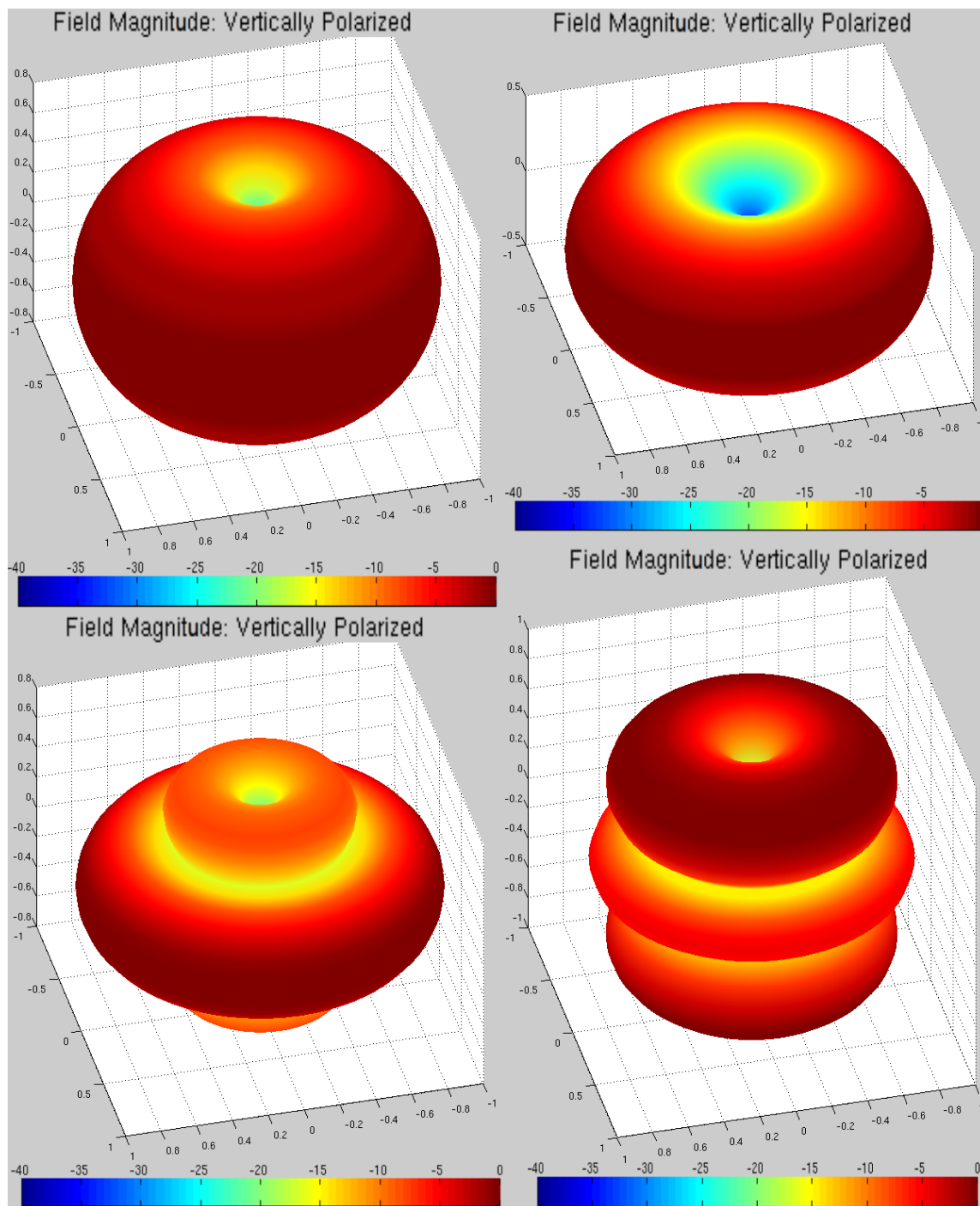


Figure 4.23: Radiation Patterns for $\lambda/2$, λ , $5\lambda/4$, and $3\lambda/2$ Dipoles
(clockwise from top-left)

4.2.2 Yagi-Uda Antenna

The Yagi-Uda antenna [11] consists of an array of linear dipoles. One dipole is driven with a source current and the other dipole elements serve to direct the beam. The element opposite the intended direction of radiation is longer than the driven element and serves as a reflector. The elements in the intended direction of radiation are shorter than the driven element and serve as directors [12]. Table 4.1 give the optimal array dimensions for a six-element Yagi-Uda array after spacing and length perturbation as

Length		Spacing and Radius	
l_1/λ	0.472	s_{21}/λ	0.250
l_2/λ	0.452	s_{32}/λ	0.289
l_3/λ	0.436	s_{43}/λ	0.406
l_4/λ	0.430	s_{54}/λ	0.323
l_5/λ	0.434	s_{65}/λ	0.422
l_6/λ	0.430	r/λ	0.003369

Table 4.1: Yagi-Uda Dimensions

developed by Chen and Cheng [13]. To build the Yagi-Uda antenna described in Table 4.1 six wire elements need to be created. The array will lie in the xz-plane, be directed in the positive x-direction, and be driven by a source current placed at the origin. Therefore, the reflector element will have a negative x-value, the driven element will be located at $x = 0$, and the director elements will have

positive x-values. Each element will be modeled using a wire centered on the x-axis and will have a radius of 0.003369λ and a unit resistance of 0Ω . The coordinate values of the start and stop vertices, the wire radius, `rpl`, and element name are shown below in Table 4.2. For reasons explained below, all wires are created with `ideal_del = 0.01\lambda` and `force_odd = true`. Similar results can be achieved by producing each element centered at the origin and then translating each element along the x-axis by the distance specified in column 2 of Table 4.2. This would effectively place each element in the same location as the first procedure, but would require that each component be saved via Overwrite (a step that can be quite lengthy for large geometries). Regardless of the procedure used, once all the elements are placed appropriately and saved via “Overwrite” as required, the geometries need to all be

Element	Start Vertex			Stop Vertex			Wire	Resistance
Name	X	Y	Z	X	Y	Z	Radius	Per Length
reflector1	-0.250	0	0.236	-0.250	0	-0.236	0.003369 λ	0 Ω
driven2	0	0	0.226	0	0	-0.226	0.003369 λ	0 Ω
director3	0.289	0	0.218	0.289	0	-0.218	0.003369 λ	0 Ω
director4	0.695	0	0.215	0.695	0	-0.215	0.003369 λ	0 Ω
director5	1.018	0	0.217	1.018	0	-0.217	0.003369 λ	0 Ω
director6	1.440	0	0.215	1.440	0	-0.215	0.003369 λ	0 Ω

Table 4.2: Component Geometry Parameters

loaded to the workspace and combined by pressing the “Combine” button. For this example the elements were saved to yagi_uda_6element, the workspace was reset, and geometry yagi_uda_6element was loaded by itself. Figure 4.24 shows the spacing and dipole lengths of the resulting geometry. To create the radiation pattern, the radi-

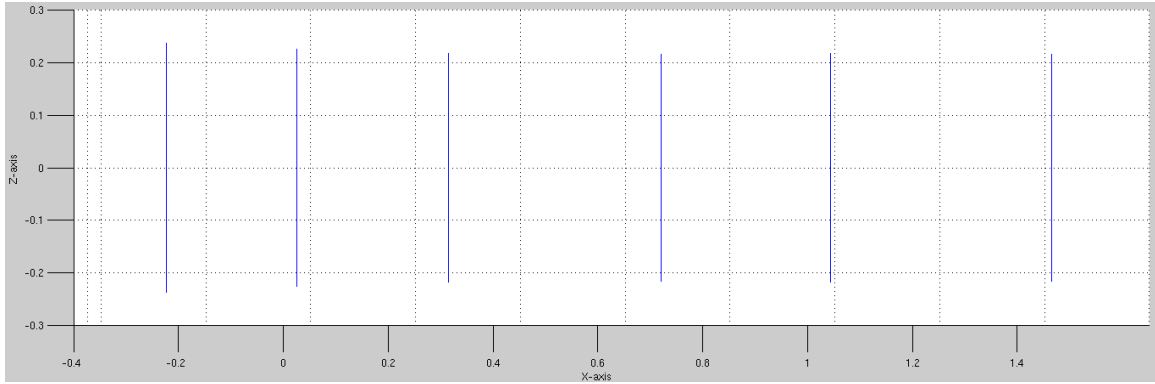


Figure 4.24: Yagi Array Geometry

ation modeler is used to place a delta source on the segment nearest the origin. To achieve accurate model output it is necessary to insure the selected segment is placed uniformly over the x-axis. If wire elements are created with force_odd = false it is possible to have a vertex lying directly on x-axis. As a result, the source current would be placed on a segment that is offset from the x-axis resulting in an inaccurate model. To remedy this the vertex at (0,0,0) would need to be destroyed using the remover and three new segments created, one of which extends equal directions in both positive and negative z to insure the source current is truly placed at the center of the driven element. By selecting force_odd = true, this step can be skipped. The

surface current plot in Figure 4.25 shows the surface currents created by a voltage source placed at the origin. Surface currents are strongest on the driven element and first director and show a marked reduction on the reflector element and subsequent directors. Radiation patterns for the yagi-uda antenna are shown in Figure 4.26 in both dB and linear scale. Both patterns show good directivity in the main lobe, but larger than expected backlobes when compared with results obtained by Balanis [12].

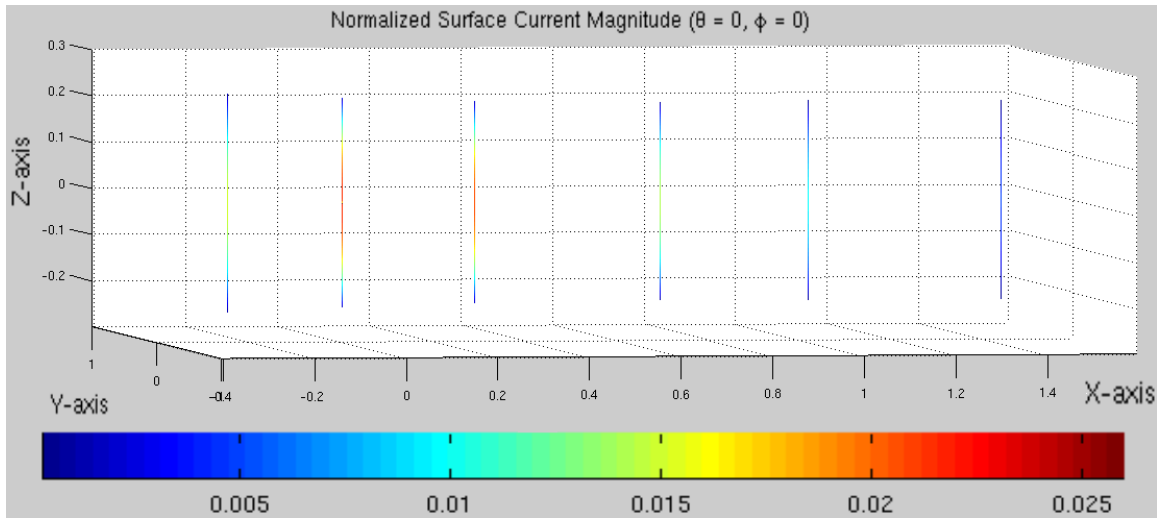
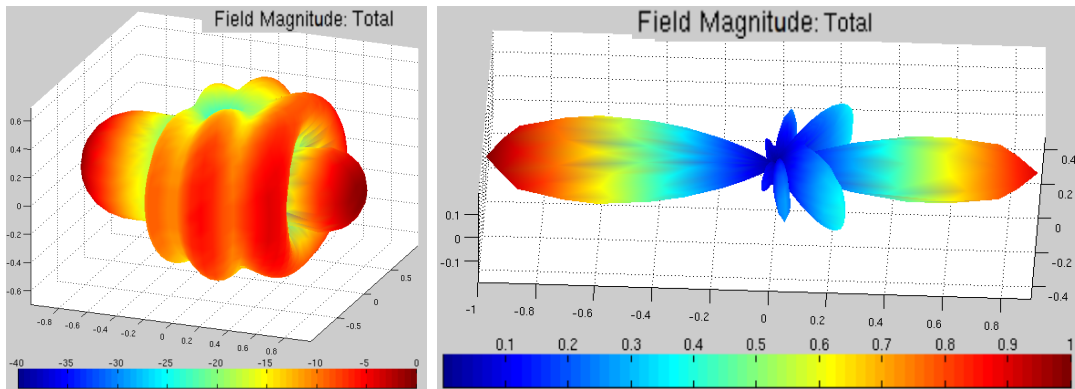


Figure 4.25: Surface Currents Due to Source at Origin



(a) dB

(b) Absolute

Figure 4.26: Radiation Pattern

4.2.3 Dipole In Spherical Cavity

Cavities are often used for testing of radiating structures. The first geometry is a full-wave dipole created with a maximum element length of $\text{ideal_del} = .01\lambda$. The start vertex is located at $(0,0,.5)$ and the end vertex is located at $(0,0,-.5)$. The radius is $.003\lambda$ and the resistance per length is 0Ω . The second geometry is a spherical cavity created with a maximum element length of $\text{ideal_del} = 0.1\lambda$ and radius $= 2\lambda$ centered at the origin. A third geometry is a cubic cavity centered at the origin created with a maximum element length of $\text{ideal_del} = 0.1\lambda$ and dimensions $= 2\lambda$. No modification is required as all elements are centered on the origin. Both geometries were saved via overwrite and then the dipole was combined with the spherical cavity to form FWDIP_SPHCAV and with the cubic cavity to form FWDIP_CUBCAV. The delta source modeler is used to place the current source on the segment closest to the origin, which is expected to be the center of the dipole. This will allow the dipole to radiate as intended within the spherical and cubic cavities. Model output appears typical until convergence is attempted during iterative solve. After 280 iterations the error was still two orders of magnitude greater than the specified target. During advisement it was explained achieving convergence within an unloaded PEC cavity was unrealistic and that the model results were inaccurate due to the poor system conditioning. However, despite the undesirable results the software shows that the possibilities for system realization are great enough to allow for even poorly modeled systems.

4.2.4 Helical Antenna

The helical antenna is comprised of a long wire coiled into a helix and attached to a ground plane. Studies show that a helical antenna shows good end-fire radiation characteristics when the circumference of the helix is roughly equal to λ . Balanis provides an easy-to-follow example where a 10-loop helical antenna is designed for end-fire radiation. The design parameters are shown below in Table 4.3 and will be used to build the helical antenna to be modeled using the GUI [12]. Good end-fire performance

Circumference	λ
Spacing	0.231λ
Loops	10
Diameter	$1/\pi$

Table 4.3: Helix Parameters

is achieved when the ground plane is larger than $\lambda/2$, helix circumference is roughly equal to λ , and spacing is near $\lambda/4$. The first element to be created is the ground plane. It is built at the origin with $l = w = \lambda$ and $ideal_del = 0.05\lambda$. No helix geometry is available in the builder GUI, so a custom MATLAB function named `buildhelix.m` was created. The `buildhelix.m` function accepts arguments specifying the `ideal_del`, loop diameter, spacing between loops, number of loops, wire radius, wire resistance per λ , and path to the geometry directory and produces a corresponding helix extending from the origin in the $+z$ direction. With both geometries loaded to the workspace, the user can choose to directly connect the two by moving the end of the helix to a common vertex on the plate and running `geoconvert.m` to create a common `vertex.dat` file. The rebuilt geo file would only use Cartesian coordinate representation of the vertices so the difference in nodal indices would not be an issue. Another option that can be employed when geometry vertices do not line up nicely is to use an intermediate piece of wire to join the two geometries. This small wire can then be resistively loaded to model the source impedance. To achieve this, the helix is first translated .001 in the $+z$ direction and the joiner used to connect the two vertices identified by their `data_cursor` information. In this example, the first method will be employed along

with a secondary method to verify vertex commonality in the final geometry. The two vertices to be connected are located at $(0, -0.15, 0)$ on the plate and $(0, -0.1592, 0)$ on the helix. In instances like this where decimal error can be a factor, it is sometimes simplest to make a ballpark translation and then edit the `elms.line.dat` file directly. Figure 4.27 shows the Cartesian coordinates of the vertices before and after translation. After translating the helix by $z = 0.0092\lambda$ the resultant geometries show a

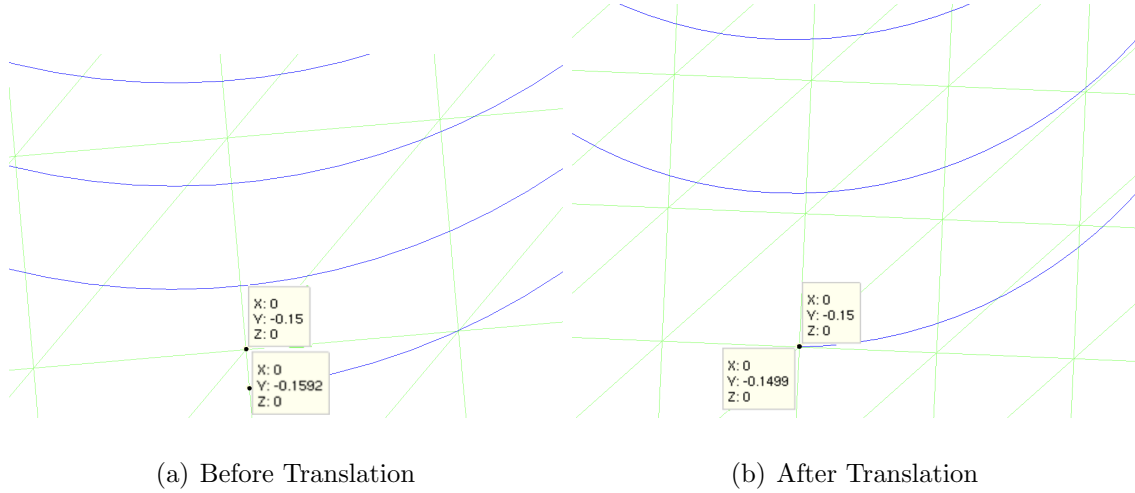
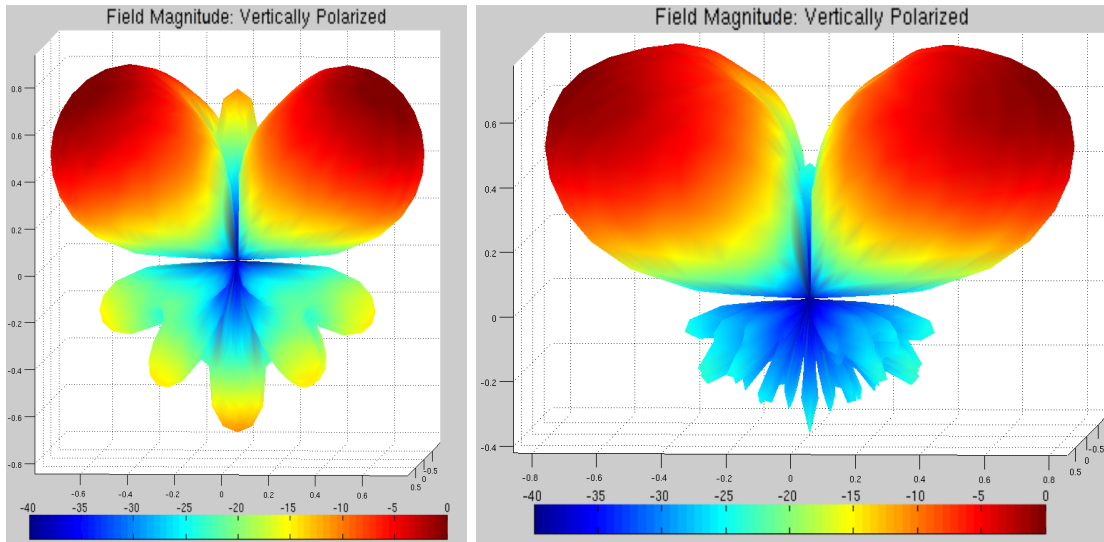


Figure 4.27: Vertex Coordinates to Combine

persistent gap of 0.0001λ . Although small, this discrepancy is large enough to allow the existence of two vertices right next to each other. To remedy this, the reference to the desired vertex at $(0, -0.15, 0)$ is located and copied to the corresponding location in `elms.line.dat`. The first step is to create a new geometry through combination, in this case the geometry is named `helix_plate` for simplicity. Investigation indicates that the desired vertex is located on line 742 of the `vertex.dat` file and the unwanted vertex is located at line 743. Simply replacing the reference to vertex 743 with a reference to vertex 742 removes the discrepancy. At this point the geometry can be saved via overwrite in preparation for modeling. A delta source model with a voltage source at $(0, -0.15, 0)$ is selected. Unfortunately, the connection between the wire segment and the plate triangular element creates the same modeling failures as were created by the hemispheric dipole in Section 4.1.4

4.2.5 Dipole Over Finite Ground Plane

A single dipole geometry was used in each of the models. The half-wave dipole was initially created between the vertices $(-0.25,0,0)$ and $(0.25,0,0)$ with an ideal segment size of $\text{ideal_del} = .01\lambda$. The wire radius and rpl were set to $.003\lambda$ and 0Ω , respectively. Two plate geometries were created using an ideal element size of $\text{ideal_del} = 0.1\lambda$ and were centered at the origin. The first plate was $2\lambda \times 2\lambda$ and the second plate was $5\lambda \times 5\lambda$. Four final geometries were created that varied in ground-plane size and dipole height. The first and second geometries had their dipole placed $h = 0.5\lambda$ above the center of the ground plane. The third and fourth geometries had their dipole placed $h = 1.0\lambda$ above the center of the ground plane. The first and third geometry used a ground-plane size of $2\lambda \times 2\lambda$ and the second and fourth geometry used a ground-plane size of $5\lambda \times 5\lambda$. The system was modeled using the delta source model with the current source placed on the segment nearest $(0,0,0.5)$ for the first two dipoles and nearest $(0,0,1.0)$ for the second two dipoles. Each dipole exhibited the expected

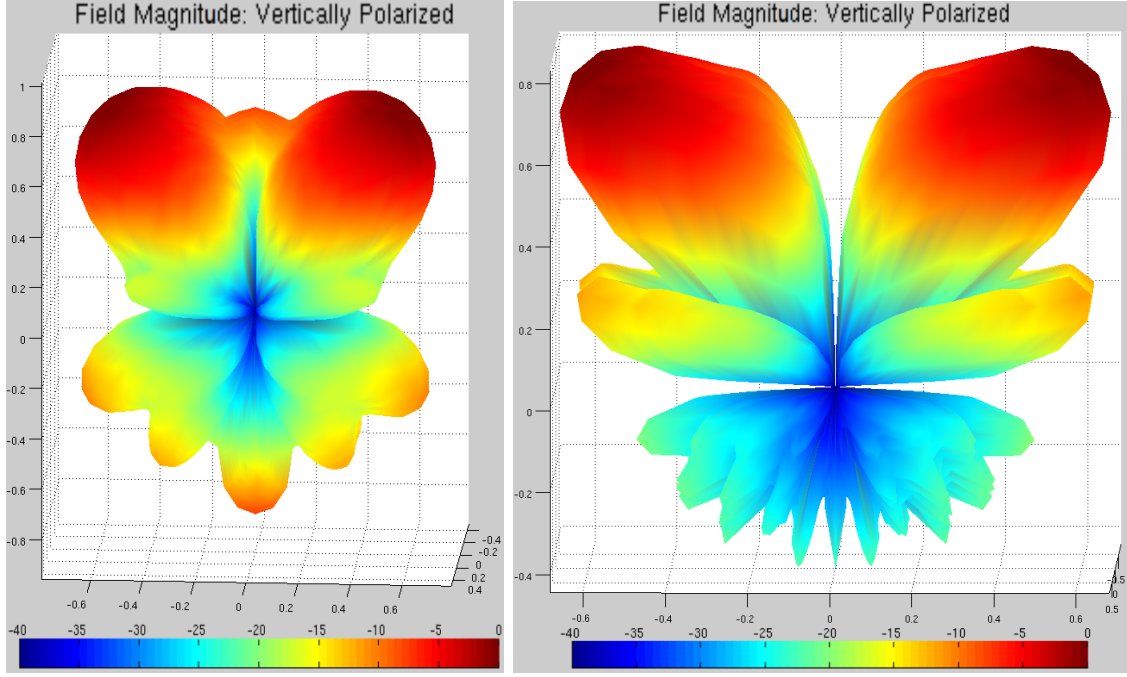


(a) Over $2\lambda \times 2\lambda$ Plate

(b) Over $5\lambda \times 5\lambda$ Plate

Figure 4.28: Halfwave Dipole at Height of $\lambda/2$

sinusoidal current distribution radiation from the center. The radiation patterns of the dipole over the smaller plate were nowhere near the results obtained by Kraus



(a) Over $2\lambda \times 2\lambda$ Plate

(b) Over $5\lambda \times 5\lambda$ Plate

Figure 4.29: Halfwave Dipole at Height of λ

and Marhefka for a half-wave dipole over an infinite ground-plane [10]. However, the dipoles over the larger ground-planes showed that the radiation pattern was trending toward the theoretical pattern for a dipole over an infinite ground plane. Figures 4.28 and 4.29 show the resultant radiation patterns for a dipole at height $h = 0.5\lambda$ and $h = \lambda$ above a plate, respectively. Increasing the size of the plate provides a noticeable reduction in the size of the backlobes as well as better development of the sidelobes as energy is prevented from reaching the backside of the plate and instead is re-radiated in the direction of the dipole. As the size of the plate grows it can be expected that the radiation pattern will continue to converge with the theoretical radiation pattern for a dipole over an infinite plate. Figure 4.30 displays the surface currents resulting from a half-wave dipole at a height of $\lambda/2$ above a $5\lambda \times 5\lambda$ plate.

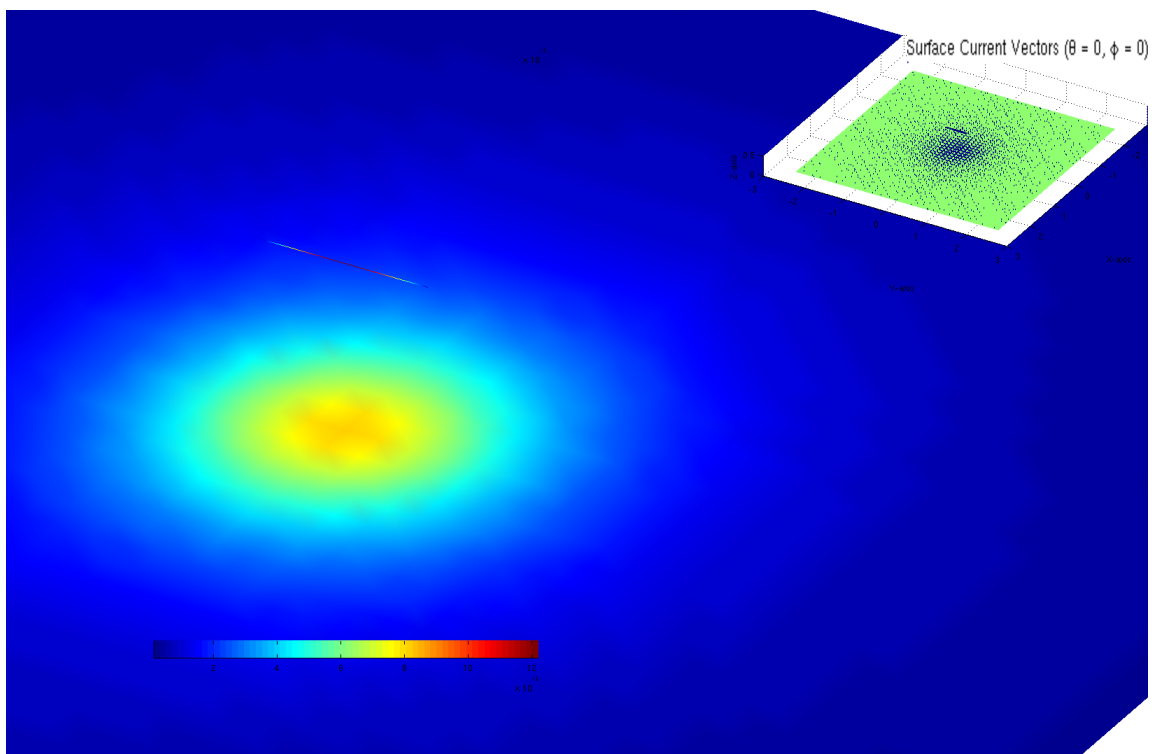


Figure 4.30: Surface Currents for Dipole Height of $\lambda/2$

CHAPTER 5

CONCLUSIONS

The primary purpose of this thesis was to make the computation modeling software developed by RAFTAS more approachable. The main challenges to overcome were the limited avenues for geometry creation, code-intensive modeling procedures, significant user learning curve, unsorted approach to data storage, cursory presentation of data, and limitations in work locale. Each of the main challenges identified in Table 1.1 is addressed below and a before-and-after look provided in each area.

Facilitate Creation of Custom Geometries

The software originally had 89 test cases available to users. Although plenty of pre-coded structures existed, the opportunity for user customization was typically limited to pre-defined allowable variances in dimensional tolerances. All geometry creation was accomplished inside C++ executables with no readily available method for combination of test cases. To create new geometry options a user would need to correctly produce a C++ executable to accomplish their goals.

By comparison, the MLFMA GUI only offers six pre-loaded geometries. However, every aspect of these geometries can be customized to accommodate specific modeling requirements and geometries can now be combined and arranged relative to other geometries in 3D space. All geometry creation is accomplished within a software function as before; however, the resultant components of the geometry are now available for manipulation within the MLFMA GUI environment as well as within the MATLAB workspace. Within the MATLAB workspace, for example, the astute user

can introduce roughness by nodal introduction of radial and planar randomness using the MATLAB `rand()` function. And finally, the opportunity for expansion of the geometry build library is facilitated by ease of function creation afforded by MATLAB. Geometry creation functions can produce Delaunay geometry files by adding the six lines of code below, as required, where `ppath` is the hyphen-terminated path to the desired geometry folder.

```
unix(['mkdir ',ppath]);
dlmwrite([ppath,'vertex.dat'],pts);
dlmwrite([ppath,'elms_line.dat'],line);
dlmwrite([ppath,'elms_tri.dat'],tri);
dlmwrite([ppath,'elms_quad.dat'],quad);
georevert(ppath);
```

Increase Approachability of Modeling Capabilities

Creating a model using the original REFTAS software required either, at a minimum, an understanding of the source, solver, preconditioner, and system parameters or a copy of the source code used to perform the desired modeling on a similar geometry. To understand what form of modeling was taking place within a given file, the source code had to be opened and investigated for calls to source creation function.

The MLFMA GUI provides three standardized modeling options: backscatter, delta source, or Hertzian dipole. The procedure is already provided and default model parameters already provided for each type of modeling. The only options available to the astute coder that are not provided in the MLFMA GUI are those that were overlooked and can easily be incorporated into a future revision of the GUI.

Reduce Learning Curve Required for Proficiency

Interfacing with the original software executables took place at the command line. Test case files had to be called from the directory they were stored in and variable input was provided by C++ cin and cout functions. For anyone but the original creator to understand the capabilities of the source code, they would need to open it and read the author's comments, if any were provided. Creation of executables from the command line also required a debugging and trial and error.

By its very nature, the MLFMA GUI provides visual clues as to what each button, dropdown box, and file list relate to. This prevents the cause-and-effect focuses learning curve present in command line interaction. Additionally, the GUI maintains a list of pathways to relevant geometry files in the GUI workspace to limit the user's need to dig through the actual folder hierarchy in search of data. Lastly, trial and error is limited to the creation of geometries within the MATLAB workspace and function creation environment. All modeling algorithms are standardized and methods of interaction limited to provision of interface geometry files and selection of model parameters within the model GUIs.

Expand Data Storage and File Retrieval Options

The standard output was the only form of data stored using the original software. The information contained in the standard out was the computation progression of the modeler as it allocated memory, performed linear algebra, and pursued convergence. The culmination of which consisted of a table of field magnitudes observed at pre-determined observation angles. Retrieval of these files allowed insight into the radiation and scattering characteristics of the model, but did not preserve any data concerning the patches and wires that constituted the geometry nor any information about the surface currents present on those patches and wires.

The MLFMA continues to store the standard output in the form of model logs. However, the field magnitudes stored at the end of the standard output are segregated and post-processed to produce easily assimilated data files. Additionally, the patch and wire data used to build the original geometry is available for reference in the model directory and also available in the geometry folder for follow-on models based on the same geometry. Lastly, the surface currents present at each vertex are retrieved, segregated and post processed for easy assimilation by the surface current viewer.

Improve Data Presentation

Data produced by the command line software was presented in the form of a table at the end of the standard out and a command-line call to the BAMG function provided mesh visualization, however that was the extent of the graphical presentation of the data. The MLFMA GUI has three visualization options. The geometry viewer provides a mesh representation of the patches elements and wire segments in 3D space. The surface current viewer allows the surface currents induced by each source to be drawn. And the field pattern viewer provides visual interpretation of the field magnitudes provided by the original standard out.

Remove Access Limitations

Originally, all software was hosted on REFTAS workstations. To have access to the software, users had to either be on location or have an active SSH connection to a REFTAS workstation. Despite the method of access, interaction was conducted through the command line.

Now, the software is portable. Reducing the reliance on custom implementation files and only carrying the executables means that the entire MLFMA directory takes up less than 10 MB of disk space; a size requirement easily satisfied by even the oldest of usb thumb drives. The only user limitations are the requirement that a Linux OS

be used and a MATLAB installation is accessible. However, further development of GUI file handling and installation of `ispc()` and `isunix()` switch statements whenever commands are executed from the command line could easily remove the OS limitation. Additionally, conversion of the GUI MATLAB files to mex files would allow the software to operate outside the presence of a MATLAB installation.

BIBLIOGRAPHY

- [1] R. F. Harrington, *Time-Harmonic Electromagnetic Fields*. John Wiley and Sons, Inc, 2001.
- [2] C. W. Steele, *Numerical Computation of Electric and Magnetic Fields*. Chapman Hall, 2nd ed., 1997.
- [3] J. L. Volakis and D. B. Davidson, “Evaluation of the bicgstab(l) algorithm for the finite-element/boundary-integral method,” *IEEE Antennas and Propagation Magazine*, vol. 43, no. 6, pp. 124–131, 2001.
- [4] J. Lee, J. Zhang, and C.-C. Lu, “Performance of preconditioned krylov iterative methods for solving hybrid integral equations in electromagnetics,” Tech. Rep. 373-03, Department of Computer Science, University of Kentucky, 2003.
- [5] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. V. der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, 2nd ed., 1994.
- [6] MATLAB, *version 7.10.0 (R2010a)*. Natick, Massachusetts: The MathWorks Inc., 2010.
- [7] R. Salisbury, “gtools v1.0 documentation.” gtools Package for MLFMA, 2008.
- [8] S. Holz, “cline.” (<http://www.mathworks.com/matlabcentral/fileexchange/14677-cline>) MATLAB Central File Exchange. Retrieved March 31st, 2011.
- [9] A. P. Engsig-Karup, “Quadmesh quadrilateral mesh plot.” (<http://www.mathworks.com/matlabcentral/fileexchange/20266-quadmesh->

- quadrilateral-mesh-plot) MATLAB Central File Exchange. Retrieved March 11th, 2011.
- [10] J. D. Kraus and R. J. Marhefka, *Antennas for all Applications*. McGraw-Hill Higher Education, 3rd ed., 2002.
 - [11] H. Yagi, “Beam transmission of ultra-short waves,” *Proceedings of the IRE*, vol. 16, pp. 715–741, June 1928.
 - [12] C. A. Balanis, *Antenna Thoery*. Wiley Interscience, 3rd ed., 2005.
 - [13] C. A. Chen and D. K. Cheng, “Optimum element lengths for yagi-uda arrays,” *IEEE Transactions Antennas and Propagation*, vol. AP-23, pp. 8–15, January 1975.
 - [14] L. Lamport, *L^AT_EX User’s Guide and Reference Manual*. Addison Wesley, 2nd ed., 1994.
 - [15] C. A. Balanis, *Advanced Engineering Electromagnetics*. John Wiley and Sons, Inc, 1st ed., 1989.
 - [16] J. C. West and C. Bunting, “Modeling of lossy, enclosed rooms using mlfma,” *Proceedings of the 2009 IEEE Antennas and Propagation Society International Symposium*, June 2009.
 - [17] F. Hecht, “Bamg: Bidimensional anisotropic mesh generator,” tech. rep., University Pierre et Marie Curie, 2006.

VITA

Christopher Carrero

Candidate for the Degree of
Master of Science

Thesis: MATLAB GEOMETRY BUILDER AND MLFMA MODELER

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in Bend, OR, USA on January 25th, 1981.

Education:

Received the B.S. degree from Oklahoma Christian University, Edmond, OK, USA, 2004, in Electrical Engineering

Received the Master of Business Administration from Oklahoma Christian University, Edmond, OK, USA, 2005.

Completed the requirements for the degree of Master of Science with a major in Electrical Engineering Oklahoma State University in July, 2011.

Experience:

Lifecycle Sustainment Engineer with the Department of Defense, United States Air Force, Tinker AFB.

Name: Christopher Carrero

Date of Degree: July, 2011

Institution: Oklahoma State University

Location: Stillwater, Oklahoma

Title of Study: MATLAB GEOMETRY BUILDER AND MLFMA MODELER

Pages in Study: 81

Candidate for the Degree of Master of Science

Major Field: Electrical Engineering

Development of MATLAB graphical user interface to facilitate building and modification of discretized geometries for use in moment method and multilevel fast multipole algorithm modeling and graphical depiction of resultant geometry, surface currents, and field magnitudes. MM and MLFMA software developed at Oklahoma State University Robust Electromagnetic Field Testing and Simulation (REFTAS) Laboratory.

ADVISOR'S APPROVAL: _____